HUMBOLDT-UNIVERSITÄT ZU BERLIN

INSTITUT FÜR BIBLIOTHEKS- UND INFORMATIONSWISSENSCHAFT

BERLINER HANDREICHUNGEN
ZUR BIBLIOTHEKS- UND
INFORMATIONSWISSENSCHAFT

HEFT 413

ENTWICKLUNG UND EVALUATION EINES NEUEN WEBSERVICE MIT DEN RE3DATA.ORG-METADATEN ZUR VERBESSERUNG DER USABILITY DER RE3DATA.ORG-DIENSTLEISTUNGEN

VON

FLORIAN FRITZE

Entwicklung und Evaluation eines neuen Webservice mit den re3data.org-Metadaten zur Verbesserung der Usability der re3data.org-Dienstleistungen

von
Florian Fritze

Berliner Handreichungen zur
Bibliotheks- und Informationswissenschaft

Begründet von Peter Zahn
Herausgegeben von
Konrad Umlauf
Humboldt-Universität zu Berlin

Heft 413

Fritze, Florian

Entwicklung und Evaluation eines neuen Webservice mit den re3data.org-Metadaten zur Verbesserung der Usability der re3data.org-Dienstleistungen / von Florian Fritze. - Berlin : Institut für Bibliotheks- und Informationswissenschaft der Humboldt-Universität zu Berlin, 2016. - 70, 5, 9 S. : graph. Darst. - (Berliner Handreichungen zur Bibliotheks- und Informationswissenschaft ; 413)

ISSN 14 38-76 62

Abstract:

Der Webservice re3data.org hat sich als erste Anlaufstelle für die Suche nach geeigneten Forschungsdatenrepositorien etabliert. Das Ziel dieser Arbeit ist es, mit den frei verfügbaren Metadaten von re3data.org eine Webseite mit Datenbankanbindung zu erstellen, die die Usability der Webseite im Vergleich zur offiziellen Version verbessert. Dabei werden unterschiedliche Technologien zur Realisierung eines Webservice dargestellt und z. T. auch praktisch angewendet. Der dritte Teil der Arbeit beinhaltet zwei vergleichende Usability-Evaluationen, die das Design, sowohl der offiziellen re3data.org-Webseite als auch der in dieser Arbeit erstellten, bewerten sollen. Damit soll untersucht werden, ob das Ziel, die Verbesserung der Usability durch eine Neuimplementierung, erreicht wurde.

Diese Veröffentlichung geht zurück auf eine Masterarbeit im weiterbildenden Masterstudiengang im Direktstudium Bibliotheks- und Informationswissenschaft (Library and Information Science, M. A. (LIS)) an der Humboldt- Universität zu Berlin.

Online-Version: http://edoc.hu-berlin.de/series/berliner-handreichungen/2017-413

Ich bedanke mich bei den folgenden Menschen für ihre direkte oder indirekte Unterstützung:
Ernst Schöttle, Jörg Hinrichs, Frank Hirsch, Christhard-Georg Neubert, Maxi Kindling und Stephanie van de Sandt

1 Inhalt

Einleitung

Diese Masterarbeit gibt eine Antwort auf die Forschungsfrage, wie ein Bibliothekar oder Webdesigner die Usability eines Webservice erhöhen kann, indem er diesen Service neu implementiert. Unter Neuimplementierung verstehe ich nicht nur eine Veränderung des Layouts der Webseite, sondern auch eine Veränderung der Technologien, die das Fundament der Webseite bilden: z.B. die Wahl eines anderen Webservers oder einer anderen Datenbanktechnologie. Diese Veränderungen am Fundament und an der Oberfläche sollen zu einer Verbesserung der Usability führen, womit sich die Frage der Messbarkeit dieser Veränderungen an der (Web-) Oberfläche und damit der Usability Evaluation stellt. Bevor ich zu einer Präzisierung des Themas und meiner Intention hinter der Forschungsfrage komme, möchte ich die Gliederung dieser Arbeit vorstellen.

1.1 Gliederung der Masterarbeit

Das erste Kapitel umfasst die Einleitung, die Präzisierung und Begründung des Themas sowie den Literaturbericht. Im zweiten Kapitel im ersten Unterkapitel stelle ich mit Hilfe von UML-Diagrammen und Codebeispielen meine Implementierung des Webservice vor. Darauffolgend werfe ich im zweiten Unterkapitel einen Blick auf Programmiersprachen-Rankings, um in den folgenden drei Unterkapiteln drei weitere Technologien zu erläutern, mit denen ein Webservice implementiert werden kann. Das dritte Kapitel beinhaltet die vergleichende Evaluierung der Usability zweier Implementierungen eines Webservice. Im vierten Kapitel werde ich meine Forschungsfrage abschließend beantworten.

1.2 Präzisierung und Begründung des Themas

Die Idee für meine Forschungsfrage kam mir bei meiner Arbeit als studentische Hilfskraft im DFG-geförderten Projekt re3data.org. Dieses Projekt ist ein globales Verzeichnis von Forschungsdatenrepositorien unterschiedlicher Disziplinen. Projektpartner in re3data.org sind die Abteilung für Bibliotheks- und Informationsdienstleistungen des Geoforschungszentrums in Potsdam, die Bibliothek des KIT in Karlsruhe, die Bibliotheken der Purdue Universität in den USA und das Institut für Bibliotheks- und Informationswissenschaft der Humboldt-Universität zu Berlin (Rücknagel 2015, Vgl. S. 2), welche mein Arbeitgeber war. Meine Arbeit bestand darin Forschungsdatenrepositorien zu erschließen. Diese Repositorien spielen eine entscheidende Rolle im wachsenden Bereich des Forschungsdatenmanagements. Ein Repositorium, das potentiell in re3data.org aufgenommen werden könnte, hat folgende Eigenschaften: Es hat einen Fokus auf Forschungsdaten, es wird von einer juristischen Person innerhalb eines organisatorischen Rahmens betrieben, der Nachhaltigkeit unterstützt (wie z. B. eine Bibliothek oder Universität), es setzt klare Zugriffsbedingungen zum Repositorium und den dort gespeicherten Forschungsdaten und stellt Nutzungsbedingungen bereit (Rücknagel 2015, Vgl. S. 4). Die Metadaten zu einem Repositorium wurden in Form einer XML-Datei an das KIT geschickt, damit der zuständige Informatiker diese neuen Metadaten in die Datenbank von re3data.org einspielen und damit für jeden über die Webseite des Projekts zugänglich

machen konnte. Da ich durch die Arbeit als Erschließer mit dem Metadatenschema gut vertraut bin, war mir aufgefallen, dass viele Informationen zu einem Repositorium über die Projektseite (www.re3data.org) nicht explizit als Suchanfrage ausgewählt werden konnten. Dies mag sich mit dem Release der neuen Oberfläche im Frühjahr 2016 geändert haben. Jedoch kam mir im Laufe des vergangenen Jahres 2015 die Idee, die Webseite anzupassen, um bessere Suchmöglichkeiten für die Nutzer von re3data.org anbieten zu können. Diese Idee, einen neuen bzw. angepassten Webservice zu implementieren, wurde noch konkreter als am 9. März 2015 eine REST-API vom Projekt veröffentlicht wurde, mit der jeder Nutzer per HTTP-GET-Request den vollen Zugriff auf alle Metadaten zu den Repositorien hat, die in re3data.org verzeichnet sind. Diese Metadaten sind das Herzstück von re3data.org und da diese zum Download bereit stehen, ist es mir möglich, eine Datenbank mit ihnen zu füllen und eine Webseite zu schreiben, die auf die Metadaten zugreift, um diese, so meine Intention, besser durchsuchbar zu machen. Im Folgenden möchte ich im Literaturbericht auf die theoretischen und technischen Grundlagen meiner Arbeit näher eingehen.

1.3 Literaturbericht

Die veröffentlichten Metadaten müssen, damit sie leicht abfragbar sind, in eine Datenbank geschrieben werden. Beginnen möchte ich den Literaturbericht daher mit dem technischen Kontext meiner Arbeit und der von mir eingesetzten Datenbanktechnologie: Für meine Implementierung habe ich als Datenbank einen Sesame-Triplestore ausgewählt, in dem die Metadaten als RDF-Daten vorliegen. Sesame ist ein Open Source Java-Framework, um RDF-Daten zu verarbeiten (Vgl. Sesame 2016). Diese RDF-Daten werden in der Regel in einer speziellen Datenbank namens Triplestore gespeichert. RDF bedeutet „Resource Description Framework“ und „ist eine formale Sprache für die Beschreibung strukturierter Informationen (…) [und] wird oft als grundlegendes Darstellungsformat für die Entwicklung des Semantic Web angesehen.“ (Hitzler et al. 2007, S. 35) DuCharme definiert das Semantic Web „als ein Satz von Standards und Best-Practice-Lösungen, um Daten und ihre Semantik über das Web zu teilen und für Applikationen nutzbar zu machen.“ (DuCharme 2013, Vgl. S. 19) Theoretisch könnte ich diese RDF-Daten veröffentlichen, indem ich den Sesame-Triplestore online stelle und so konfiguriere, dass SPARQL-Anfragen, über die REST-Schnittstelle eingehen dürfen und beantwortet werden. Die SPARQL Version 1.1 ist nach der W3C-Definition ein Satz von Spezifikationen, die Sprachen und Protokolle bereitstellen, um RDF-Graphen im Web oder in einem RDF-Store abzufragen und zu manipulieren (Vgl. MIT 2013). „Representational State Transfer (abgekürzt REST) bezeichnet ein Programmierparadigma für verteilte Systeme, insbesondere für Webservices. REST ist eine Abstraktion der Struktur und des Verhaltens des World Wide Web.“ (Wikipedia 2016g) Mit dem Einsatz dieser Technologien hätte ich nebenbei auch einen Schritt in Richtung Linked Open Data getan. „Linked Open Data (LOD) bezeichnet im World Wide Web frei verfügbare Daten, die per Uniform Resource Identifier (URI) identifiziert sind und darüber direkt per HTTP abgerufen werden können und ebenfalls per URI auf andere Daten verweisen.“ (Wikipedia 2016e) Die RDF-Daten im Sesame-Triplestore wären für die Allgemeinheit zugänglich. Befüllt mit RDF-Daten wird der Sesame-Triplestore von einem Java-Programm,

das die REST-Schnittstelle abfragt, um alle Metadaten aller vorhandenen Repositorien herunterzuladen. Die Metadaten jedes Repositoriums werden als einzelne XML-Datei gespeichert, von dem Java-Programm ausgelesen und als RDF im Triplestore abgelegt. „XML ist eine Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten in Form von Textdateien." (Wikipedia 2016c) Damit habe ich eine Datenbank, die auf einem Apache Tomcat-Server läuft und die über eine REST-Schnittstelle SPARQL-Abfragen entgegen nimmt. „Apache Tomcat ist ein Open-Source-Webserver und Webcontainer, der die Spezifikation für Java Servlets und JavaServer Pages (JSP) implementiert und es damit erlaubt, in Java geschriebene Web-Anwendungen auf Servlet- beziehungsweise JSP-Basis auszuführen." (Wikipedia 2016a) Die Antwort auf eine SPARQL-Anfrage wird in meiner Implementierung als XML-kodierte Daten zurückgeliefert und von Javascript- und JQuery-Funktionen ausgewertet. Das Ergebnis wird auf der Webseite mit Hilfe von CSS und HTML dargestellt. Mit diesen Technologien und den offen zur Verfügung stehenden re3data.org-Metadaten möchte ich eine Webseite implementieren, die für den Fachbesucher mehr Suchmöglichkeiten anbietet und damit besser auf die Wünsche ihrer Zielgruppe eingehen kann. Als Zielgruppe definiere ich hauptsächlich den Forschungsdatenmanager einer Institution oder einen Bibliothekar, dessen Aufgabe es ist, den Forscher bei der Findung eines geeigneten Repositoriums zu unterstützen und beratend zur Seite zu stehen. Deswegen verwendet die Oberfläche Fachtermini, die im Bereich des Forschungsdatenmanagements und in der Informationswissenschaft weitestgehend bekannt sein sollten. Ob die angestrebte Zielgruppe tatsächlich eine Verbesserung in der Nutzung bemerken wird und damit die Ziele des von mir entwickelten Prototyps erreicht sind, muss eine Usability-Evaluation des Webservice zeigen, die den dritten Teil meiner Masterarbeit kennzeichnet. Die theoretischen Aspekte der Usability-Evaluation möchte ich im Folgenden, beginnend mit einer rhetorischen Frage näher beleuchten: Wozu sollte eine Firma oder eine Bibliothek, die einen Webservice anbietet, überhaupt eine Usability-Evaluation der Oberfläche und Funktionalität dieses Dienstes durchführen? Reicht es nicht aus, wenn der Anbieter eine Hilfeseite einrichtet, die bei Bedienungsproblemen konsultiert werden soll? Sollten sich nicht eher die Nutzer des Dienstes an die Bedienung gewöhnen und vielleicht Schulungen, die die Firma oder Bibliothek anbietet, besuchen, damit sie den Dienst bzw. das Softwareprodukt zu ihrer Zufriedenheit nutzen können? Diese Fragen bezogen auf eine Software, die der Nutzer im Laden seines Vertrauens kaufen kann, würde der Hersteller dieses Produkt schätzungsweise bis zu Beginn der Neunziger Jahre des 20. Jahrhunderts mit „Ja" beantworten. Aus dem einfachen Grund heraus, dass der Nutzer in der Regel das Produkt schon gekauft hat, bevor er es zu Hause installiert und ausprobiert hat. Der Handel ist schon zu Stande gekommen und die Firma bzw. das Geschäft, in dem die Software verkauft wird, muss sich nicht mehr um die Zufriedenheit des Kunden kümmern. Diese Sichtweise hat sich durch den Siegeszug des Internets und seine browserbasierten Anwendungen fundamental geändert. Browserbasierte Anwendungen laufen ohne Installation auf dem lokalen PC. Sie bieten dem Nutzer zu jeder Zeit und an jedem Ort Zugriff auf seine Daten und Arbeitsprozesse. Peter Schirmbacher spricht in diesem Fall auch vom „wissenschaftlichen Kommunikationsprozess im Wandel" (Schirmbacher 2014, Folie 15).

Dieser Wandel durch Anwendungen, die im Browser laufen, betrifft kommerzielle Anbieter noch viel stärker. Der Kunde verlässt die Webseite in kürzester Zeit, wenn er mit ihrer Bedienung nicht zu Recht kommt. Das bedeutet bei einem Online-Shop, dass kein Umsatz gemacht wird. Das Prinzip, dass zuerst gekauft wird und später ausprobiert wird, hat sich im Internet fundamental gewandelt. Deswegen ist es wichtig, dass eine Webseite bzw. eine browserbasierte Anwendung gut bedienbar und möglichst selbsterklärend funktioniert. Dafür testet der Webdesigner oder die Firma, die sie in Auftrag gegeben hat, ihre Usability. Um es aus einer anderen Perspektive nochmal zu wiederholen: Die Aufmerksamkeit eines Nutzers ist sehr begrenzt, findet er nicht das, was er sucht, wendet er sich ab. Michael Goldhaber spricht in einem ähnlichen Zusammenhang auch von der „Attention Economy“ (Goldhaber 1997). Da auch das Warten auf eine Webseite den Nutzer dazu verleitet diese wieder zu verlassen, sprach Jakob Nielsen in einem anderen Kontext als Goldhaber auch von „‘Attention Economics‘“ (Manhartsberger and Musil 2002, S. 29). Da sich die Aufmerksamkeit eines Nutzers zuerst auf das Benutzerinterface richtet, ist es, nach Dhillon, heute der entscheidende Faktor für den Erfolg eines Softwareprodukts (Dhillon 2004, Vgl. S. 89). Manhartsberger und Musil formulieren es so: „Erklärt sich die Anwendung nicht selbst, wird sie nicht benutzt werden.“ (Manhartsberger and Musil 2002, S. 18) Diese Feststellung gilt im Internet umso mehr. Usability-Pionier Jakob Nielsen formuliert es ebenso knapp: „‘Users know the location of the Back Button.‘“ (Manhartsberger and Musil 2002, S. 18) „Das Internet hat also offensichtlich dem Benutzer zum Durchbruch als zentralem Steuerungsfaktor für die Entwicklung von Websites verholfen. Und damit auch dem Thema Usability.“ (Manhartsberger and Musil 2002, S. 20) Mit diesem kleinen Exkurs möchte ich zeigen, dass diese Aufmerksamkeits-Prinzipien und die damit einhergehende, notwendige Usability-Evaluation auch für nicht kommerzielle Webanwendungen gelten, wie bspw. den Webservice von re3data.org. Da ich die Notwendigkeit von Usability Engineering und der damit verbundenen Usability Evaluation aufgezeigt habe, stellt sich die Frage, wie definiert die Literatur bzw. die Fachwelt den Begriff Usability und Usability Engineering? Denn ist eine Definition vorhanden, so kann ich auch Kriterien für das Konzept der Usability finden. Dhillon definiert Usability folgendermaßen: „This is the quality of an interactive system with regard to factors such as user satisfaction, ease of use, and ease of learning.“ (Dhillon 2004, S. 2) Usability ist auch eine ISO-Norm und findet sich unter der folgenden Notation DIN EN ISO9241,11: „Usability ist das Ausmaß, in dem ein Produkt durch bestimmte Nutzer in einem bestimmten Nutzungskontext genutzt werden kann, um bestimmte Ziele effektiv, effizient und zufriedenstellend zu erreichen.“ (Rahn 2010, Folie 3) Es wird deutlich, dass Usability wirtschaftliche Aspekte hat, wenn von Effizienz und Effektivität in den Definitionen gesprochen wird. Usability kann auch einen wichtigen Einfluss auf den Betrieb einer Organisation bzw. eines Unternehmens haben, denn sie führt zu verbesserter Arbeitsleistung, größerer Produktivität und geringeren Kosten ((Delone and McLean 2003) referenziert in (Wagner, Hassanein, and Head 2014, Vgl. S. 270)). Ferner sollen die Ziele, die der Nutzer mit einer Software verfolgt „zufriedenstellend“ und einfach erreicht werden können. „Usability wird manchmal im engeren Sinne als Gütekriterium für die Gestaltung einer Benutzeroberfläche verstanden.“ (Richter and Flückiger 2013, S. 4) Die Betrachtung

der Benutzeroberfläche allein greift im Hinblick auf die Usability aber oft zu kurz. Auch geht es nicht nur darum, dass Usability „ausschließlich eine Eigenschaft eines Produktes“ sei. Puscher wendet außerdem ein, dass „Usability (…) in keiner Form ein technisches Thema“ sei. Sie sei „vielmehr ein soziopsychologisches Problem in der Wahrnehmung und Benutzung interaktiver Medien“ (Puscher 2001, S. 5). Es ist eher so, dass „Usability (…) die Eignung eines Systems [umfasst], die Abläufe und Prozesse aus Benutzersicht zu unterstützen.“ (Richter and Flückiger 2013, Vgl. S. 5-6) Bei dieser Unterstützung sollte es auch immer um die Einfachheit der Bedienung und um die Vermeidung von Unannehmlichkeiten für die Nutzer des Softwaresystems gehen. Usability (Science) als wissenschaftliche Disziplin wird von Pearrow deswegen so definiert: „Usability is the broad discipline of applying sound scientific observation, measurement, and design principles to the creation and maintenance of Web sites to bring about the greatest ease of use, ease of learnability, amount of usefulness, and least amount of discomfort for the humans who have to use the system.” (Pearrow 2007, S. 15) Damit dies so ist und schon während der Produktentwicklung berücksichtigt werden kann, gibt es das Usability Engineering und die damit verbundene Usability Analyse und Evaluation. Der Begriff des „Usability Engineering“ kam in der Mitte der 1980er Jahre auf ((Butler 1996) referenziert in (Dhillon 2004, Vgl. S. 1)). Doch schon nach dem Ende des 2. Weltkriegs begannen Forscher sich mit dem Feld der Mensch-Maschine-Interaktion zu beschäftigen, gerade auch im Hinblick auf die zunehmende Komplexität von militärischen Systemen, die von Menschen bedient werden sollen (Dhillon 2004, Vgl. S. 1). In den 1970er Jahren wurde Computer-Experten klar, dass das Interface-Design ein wichtiger Teil des Software-Engineering-Prozesses werden würde. Psychologen begannen „sich für die Gestaltung von Dialogsystemen zu interessieren.“ (Manhartsberger and Musil 2002, S. 33) Das bessere Verständnis der Mensch-Maschine-Interaktion sollte diese Systeme schneller und effizienter machen - die Vereinfachung der Nutzung dieser Systeme stand dabei aber noch nicht im Vordergrund. Nur eine „Splittergruppe“ von Experten hatte auch die leichtere Bedienung im Sinn (Manhartsberger and Musil 2002, Vgl. S. 33). „Eine wesentliche Aufgabe von Usability Engineering ist es [daher], unnötige Komplexität zu vermeiden und die Funktionalität eines Produktes auf ein für den Benutzer ideales Minimum zu reduzieren.“ (Richter and Flückiger 2013, S. 7) Heutzutage ist das Usability Engineering ein wichtiger Teil der Produktentwicklung, gerade weil Maschinen bzw. Computer allgegenwärtig sind und aus ökonomischen, sozialen und auch militärischen Systemen nicht mehr wegzudenken sind. Dhillon bietet für den Begriff „Usability Engineering“ folgende Definition: „This is iterative design and evaluation for providing customer feedback on the usefulness and usability of a system’s/product’s design functionality throughout the development phase.“ (Dhillon 2004, S. 3) Wie an dieser Definition deutlich wird, ist das Usability Engineering ein fortlaufender Prozess, schon während der Entwicklungsphase, während der das Produkt evaluiert werden soll. Das Usability Engineering schließt die Usability Evaluation explizit mit ein, Dhillon definiert sie folgendermaßen: „This is any empirical or analytical activity directed at understanding or assessing the usability of an interactive product/system.“ (Dhillon 2004, S. 3) Evaluation eines interaktiven Systems in diesem Sinne kann also auf Erfahrungswissen (Empirie) beruhen oder auch theoretisch

durch Analyse gewonnen werden. „Mit einer Usability-Analyse kann ermittelt werden, ob und wie leicht die Ziele (…) [einer (Web-) Anwendung] vom Nutzer zu erreichen sind.“ (Broschart 2011, S. 329) Ohne eine kontinuierliche Evaluation sei es außerdem nicht möglich zu wissen, ob eine (Web-) Anwendung wahrhaft die Bedürfnisse ihrer Nutzer erfüllt (Cervone 2014, Vgl. S. 11). Da eine Software in der Regel ein interaktives System ist, bringt sie den Nutzer oder Anwender ins Spiel, auf den sich meiner Ansicht nach alles konzentrieren sollte. Laut Dhillon gibt es drei weitere wichtige Faktoren, die während der Softwareentwicklung beachtet werden sollten: Kosten, Wettbewerb und der globale Markt (Dhillon 2004, S. 90). Der Nutzer ist aber meines Erachtens für eine (kommerzielle) browserbasierte Anwendung der wichtigste. Da Menschen bzw. Nutzer ihre Umwelt und damit die verwendete Software, die sie verwenden, unterschiedlich wahrnehmen, kann es aber keine objektive Usability geben. „Usability ist relativ.“ Das Softwareprodukt kann für den einen Nutzer einfach und leicht zu bedienen sein, während es für den anderen Schwierigkeiten bereitet (Dhillon 2004, Vgl. S. 91). Letztlich könnte der Produktentwickler eine Faustregel aufstellen und sagen: Wenn ein Nutzer bestimmte Aufgaben mit der Nutzung eines Produktes erfüllen kann, dann ist es nutzbar (usable). Wenn ein Nutzer dagegen seine Aufgaben mit diesem Produkt nicht erledigen kann, ist es nicht gut nutzbar (not really usable) (Sherman 2006, Vgl. S. 3). Mit diesem Ansatz haben sich die Usability-Experten und -Forscher jedoch nicht zufrieden gegeben. Sie „haben im Laufe der Zeit für eine Reihe von Gebieten Methoden, Richtlinien und Werkzeuge entwickelt, um die anwenderfreundliche Gestaltung von Anwendungen zu erleichtern.“ Dabei verwenden sie Erkenntnisse aus der Psychologie, Anthropologie und Kulturwissenschaft. (Manhartsberger and Musil 2002, Vgl. S. 39) Diese Erkenntnisse verwenden die Forscher, um Usability Evaluationen durchzuführen und damit diese zu testen. Whitefield u. a. (Whitefield, Wilson, and Dowell 1991, S. 74) haben 1991 vier Klassen von Usability-Evaluationsmethoden entwickelt (siehe Abb. 1). Larusdottir entdeckte, dass die Anwendung unterschiedlicher Usability Evaluationsmethoden auch unterschiedliche Ergebnisse produziert (Larusdottir 2011, Vgl. S. 431). Dillon plädiert im Hinblick auf die Komplexität der Entwicklung einer (Web-) Anwendung für die Verwendung und Kombination unterschiedlicher Evaluationsmethoden, auch dann schon, wenn das Produkt sich noch in

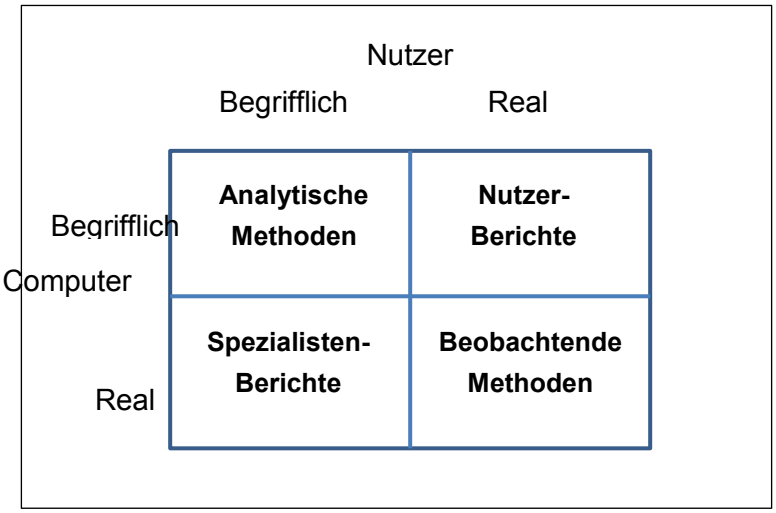

Abbildung 1: Klassen von Usability-Evaluationsmethoden

der Entwicklung befindet (Dillon 2001, Vgl. S. 1111). In der Praxis ist die Auswahl von Evaluationsmethoden nicht immer trivial und hänge von dem Ziel der Evaluation und den vorhandenen Ressourcen ab (Fu 2001, Vgl. S. 1162). Eine Nutzer-basierte Evaluationsmethode könne in der Regel nicht so viele Aspekte einer Benutzerschnittstelle abdecken wie eine Experten-basierte. Der ultimative Test sei es, wenn echte Nutzer unter normalen Arbeitsbedingungen mit dem Endprodukt arbeiten. Diesen Anspruch versuche jede Evaluationsmethode so gut es geht zu erfüllen (Dillon 2001, Vgl. S. 1111-1112). Für die Evaluation meiner Webseite werde ich jeweils eine Methode aus der Klasse der Spezialisten-Berichte und der beobachtenden Methoden einsetzen. Für die erste Methode setze ich den „BibEval"-Kriterienkatalog ein, „mit dessen Hilfe Bibliotheken selbstständig eine Usability-Evaluation ihres Webauftrittes vornehmen können und der auch von Usability-Experten als Leitfaden genutzt werden kann, um Bibliothekswebseiten effizient auf Schwachstellen überprüfen zu können." (Weinhold, Öttl, and Bekavac 2011, S. 11) Dabei wird stets der fertige Prototyp meiner Implementierung genutzt werden, um Usability-Probleme zu entdecken. Forscher teilen Usability-Probleme in die folgenden Kategorien ein: 1.) Semantische Probleme umfassen die Benennung von Features mit Begriffen bzw. Wörtern auf der Benutzeroberfläche (Volentine et al. 2015, Vgl. S. 62). 2.) Technische Probleme behandeln Navigationsprobleme und die Funktionalität der Software bzw. Webseite (Volentine et al. 2015, Vgl. S. 65). 3.) Strukturelle Usability-Probleme zeigen Schwächen bei den Schemata und Strukturen, die Inhalte organisieren und Verbindungen zwischen diesen Inhalten schaffen (Volentine et al. 2015, Vgl. S. 66). 4.) Ästhetische Usability-Probleme bzw. der visuelle Einfluss kann die User Experience beeinflussen und hat Implikationen auf die effektive Kommunikation und beeinflusst, wie der Nutzer mit der Software interagiert ((Hoffmann and Krauss 2004) und (Schenkman and Jönsson 2000) referenziert in (Volentine et al. 2015, Vgl. S. 66)). Bei der Klasse der Spezialisten-Berichte ist der Tester kein echter Nutzer der Software; er gehört also nicht der vorgegebenen Zielgruppe an. Er kann aber relevante Handbücher, Guidelines und seine eigenen Erfahrungen nutzen, um als Spezialist den Prototyp der Software direkt am Computer zu evaluieren (Fu 2001, Vgl. S. 1159). Zu dieser Klasse gehören die Experten-basierten Methoden. Dieser Begriff kann synonym für die Spezialisten-Berichte verwendet werden. Die Vorteile der Experten-basierten Methoden sind ihre Schnelligkeit und ihre geringen Kosten, da nur eine geringe Anzahl von Testern benötigt wird. Nachteile sind, dass die Variabilität bei der Beurteilung der Software durch die Experten die Ergebnisse übermäßig stark beeinflusst und dass die Tester die wahre Anzahl von Usability-Problemen überschätzen können (Dillon 2001, Vgl. S. 1111). Eine Experten-basierte Methode ist die Heuristische Evaluation. Bei dieser untersucht eine kleine Gruppe von Testern die Benutzerschnittstelle und bewertet, ob diese sich an die gängigen Usability-Prinzipien hält, die z. B. von Jakob Nielsen empfohlen werden (Fu 2001, Vgl. S. 1160). „Heuristiken sind [in diesem Zusammenhang] (Design-)Prinzipien bzw. Daumenregeln, die auf gewisse Problemgruppen bei der Gestaltung von Systemen hinweisen." Außerdem unterscheidet der Usability-Experte bei der Heuristischen und anderen Methoden der Evaluation zwischen der formativen Evaluation, die an einem sich noch in der Entwicklung befindlichen System durchgeführt wird und der summativen

Evaluation, „bei der ein voll entwickeltes System auf mögliche Usability-Schwächen untersucht wird (…)“ (Lennard and Surkau 2011, Vgl. S. 25). Eine summative Evaluation wird eingesetzt, um „nach Abschluss der Produktentwicklung (…) Urteile und Gesamteinschätzungen“ zu ermitteln ((Nielsen 1994) referenziert in (Weinhold, Öttl, and Bekavac 2011, Vgl. S. 12)). Um weitere Usability-Probleme meiner Webseite zu entdecken, verwende ich auch eine Benutzer-basierte Methode aus der Klasse der beobachtenden Methoden: den Usability-Test. „Beim Usability-Test werden Testpersonen Aufgaben gestellt, die sie auf der Website lösen sollen. (…) Die Testpersonen werden bei der Lösung der Aufgaben beobachtet und befragt, und auch bei der Befragung geht es um das Lösen der Aufgabe, die Erwartungshaltung des Benutzers und nicht um Geschmacksfragen.“ (Manhartsberger and Musil 2002, S. 324) Nach Dillon ergeben diese Benutzer-basierten Ansätze die verlässlichsten und gültigsten Einschätzungen der Usability einer Softwareanwendung. Das Ziel eines solchen Tests sei es zu untersuchen, in welchem Umfang eine Anwendung die Nutzer einer Zielgruppe bei ihrer Arbeit unterstützt (Dillon 2001, Vgl. S. 1110). „Für eine Usability-Testserie sollten Testpersonen eingeladen werden, die möglichst aus der Benutzergruppe der zu prüfenden Applikation stammen (…).“ (Richter and Flückiger 2013, S. 80-81) „Testpersonen müssen [deshalb] typische Benutzer aus der Zielgruppe sein (…), damit sie (…) den fachlichen Inhalt der Webseite verstehen.“ „Getestet werden soll ja nicht der fachliche Hintergrund, sondern wie dieser für das Web [oder eine andere Softwareapplikation] aufbereitet wurde.“ (Manhartsberger and Musil 2002, S. 320) Vor Beginn des Tests führt der Testleiter jede Testperson „in die Ziele und den Ablauf des Usability-Tests ein, und es werden Spielregeln vereinbart: Die Testperson darf den Test jederzeit unterbrechen bzw. abbrechen. Die Testperson wird meist gebeten, laut zu denken, d. h. ihre Arbeit für die Beobachter zu kommentieren. Sollte die Testperson mit einer Aufgabe nicht mehr weiterkommen, kann sie selbstständig zur nächsten Aufgabe weitergehen. Die Beobachter melden sich nur, wenn es wirklich notwendig ist. Sie sollten den Testverlauf möglichst nicht beeinflussen.“ (Richter and Flückiger 2013, S. 81) Während des Tests soll die Testperson „Standardaufgaben“ lösen, die für jede Testperson gleich sind, um Vergleichbarkeit herstellen zu können (Richter and Flückiger 2013, Vgl. S. 80). Ferner sei es das „Ziel jedes Benutzertests (…), möglichst viele Informationen über das mentale Modell des Benutzers zu erhalten. Das mentale Modell umfasst seine Erwartungshaltung und wie er die Informationen wahrnimmt (…)“, die ihm die (Web-) Anwendung präsentiert. Deshalb wird oft auf die Methode des „Thinking Aloud“ aus der Psychologie zurückgegriffen, bei der der Benutzer aufgefordert wird, „‘laut zu denken‘ und seine Eindrücke zu beschreiben.“ (Manhartsberger and Musil 2002, Vgl. S. 326) Nach Larusdottir sei die „Thinking Aloud“-Methode die effektivste Art, um Usability-Probleme zu entdecken (Larusdottir 2011, Vgl. S. 431). Über die Anzahl der Testpersonen gibt es in der Literatur unterschiedliche Angaben: Manhartsberger und Musil sind der Ansicht, dass „nicht mehr als ein Dutzend Testpersonen notwendig sind, um die Usability zu beurteilen.“ (Manhartsberger and Musil 2002, S. 319) Usability-Pionier Jakob Nielsen geht davon aus, dass sogar nur 5 Testpersonen ausreichen, um schon über 75 % der Usability-Probleme einer Software zu entdecken (Nielsen 1995). Wenn jetzt der Eindruck entstanden sein sollte, dass die Messung

von Usability direkt möglich sei, ist es stattdessen so, dass Usability nicht direkt gemessen werden kann ((Hornbæk 2006, Vgl. S. 80) referenziert in (Raza, Capretz, and Ahmed 2012, Vgl. S. 1110)), weil sie ein „subjektiver“ Stoff sei ((Çetin and Göktürk 2008) referenziert in (Raza, Capretz, and Ahmed 2012, Vgl. S. 1110)). Trotzdem ist der Anspruch der Usability-Evaluation möglichst objektive Aussagen über die Usability einer (Web-) Anwendung zu erlangen, auch wenn die Usability-Erfahrung oft nur relativ ist. Einen umfassenderen Ansatz bietet das Konzept der User Experience, die sich „mit allen Aspekten, die die Erfahrungen bei der Interaktion eines Anwenders mit einem Produkt beschreiben - sowohl subjektive als auch objektive“ - befasst. „Während sich die Usability eigentlich nur mit den technischen und objektiven Faktoren befasst, schließt die User Experience zusätzlich noch alle psychologischen und subjektiven Aspekte mit ein.“ (Broschart 2011, S. 330) „Die Prozesse zur Schaffung einer optimalen User Experience werden unter dem Begriff ‚User Centered Design‘ zusammengefasst.“ (Broschart 2011, S. 332) Damit eine (Web-) Anwendung bzw. ein Informationssystem von Endanwendern genutzt wird und sie mit dem System zufrieden sind, ist es wichtig, dass sie glauben - und das ist wieder ein subjektiver Faktor -, dass das System ihre Leistung und Produktivität erhöhen wird ((Mahmood et al. 2000, Vgl. S. 764) referenziert in (Salunke and Tuleu 2015, Vgl. S. 323)). Subjektivität führt weiter zum Begriff der Schönheit und ein weiterer, wichtiger Faktor ist die Ästhetik einer (Web-) Anwendung. „Verschiedene Arten von Messungen zeigen an, dass die Schönheit einer Webseite ein wichtiger Faktor bei der Bestimmung eines Nutzers ist, wie er die Seite erfährt und bewertet.“ (Schenkman and Jönsson 2000, Vgl. S. 375) Jordan fand außerdem für eine Anzahl von Produkten heraus, dass diese öfter genutzt werden, wenn sie dem Nutzer gefallen ((Jordan 1998, Vgl. S. 30) referenziert in (Schenkman and Jönsson 2000, Vgl. S. 367)). Deshalb sollte ein Webdesigner sowohl die Usability als auch die Ästhetik einer Webseite im Auge haben, wobei die Usability meines Erachtens einen höheren Stellenwert haben sollte, schon aus wirtschaftlichen Gründen, wie weiter oben beschrieben. Obwohl mit der Webseite von re3data.org kein Umsatz oder Gewinn gemacht wird, sollte sie nach Usability-Kriterien gestaltet sein, um die Aufmerksamkeit der Nutzer nicht zu verlieren. Denn Aufmerksamkeit ist, wie ebenfalls weiter oben beschrieben, in der digitalen Welt ein knappes Gut. Bevor ich meine Implementierung der re3data.org-Webseite nach Usability-Kriterien evaluiere, möchte ich als Nächstes die möglichen Implementierungen detailliert vorstellen.

2 Mögliche Implementierungen eines Webservice

In diesem Kapitel werde ich zunächst mit Hilfe von UML-Diagrammen meine Neuimplementierung des re3data.org-Webservice beschreiben. Im zweiten Unterkapitel werfe ich einen Blick auf das Ranking einiger der hier vorgestellten Programmiersprachen und den sich daraus ergebenden Schlussfolgerungen. Im dritten Unterkapitel werde ich die aktuelle Implementierung des offiziellen re3data.org-Webservice erläutern, die sich von meiner Implementierung durch die Verwendung einer anderen Datenbanktechnologie, des PHP-Symfony-Frameworks und der Suchfunktion Elastic-Search unterscheidet. Im vierten und fünften Unterkapitel werde ich dann noch zwei weitere unterschiedliche Technologien vorstellen, mit denen der Softwareentwickler einen Webservice aufbauen kann.

2.1 Eigenständige Implementierung eines Webservice

Meine Implementierung kennt zwei grundsätzliche, verschiedene Use-Cases und Akteure. Der eine Akteur ist der Endnutzer, in diesem Fall der Bibliothekar, der ein geeignetes Forschungsdatenrepositorium finden möchte. Der andere Akteur ist der Systemadministrator, der die RDF-Daten des Sesame-Triplestores aktualisiert. Dies wird in Abbildung 2 in der Form eines Use-Case-Diagramms dargestellt.

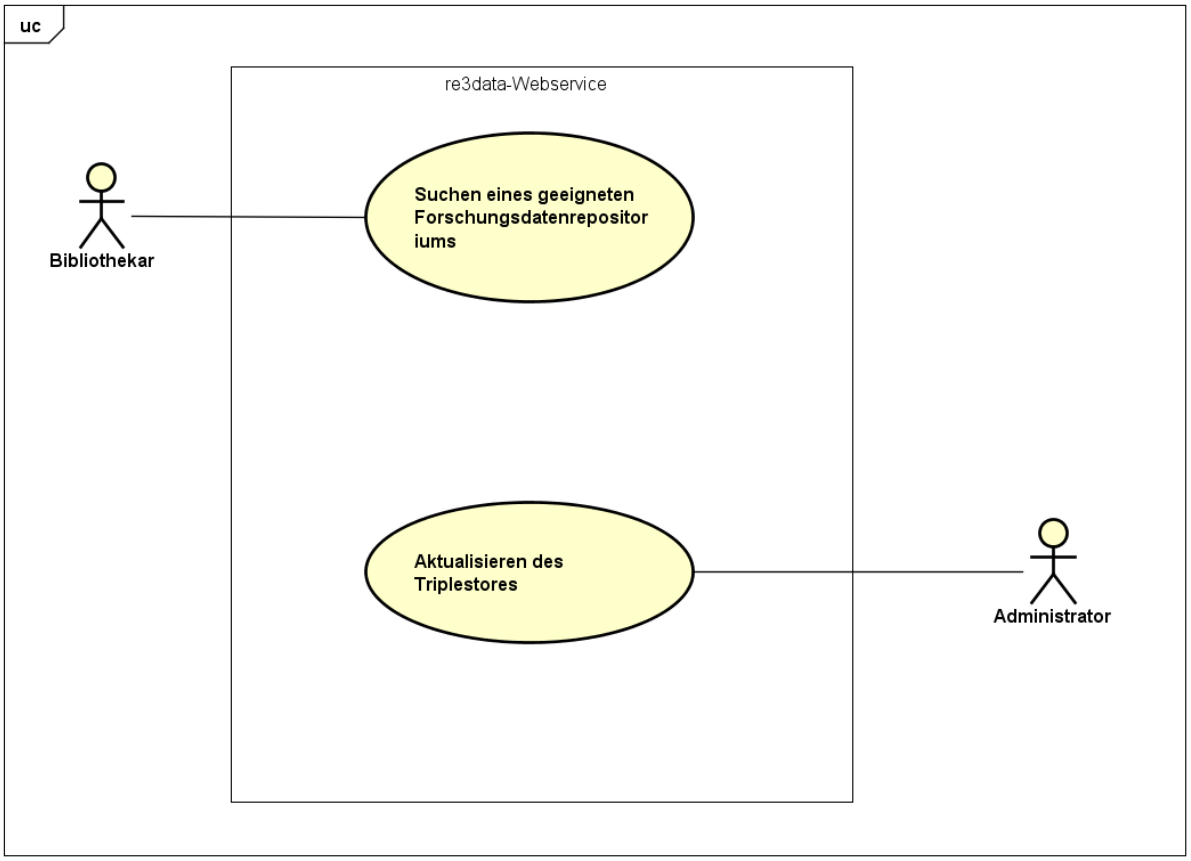

Abbildung 2: Use-Case-Diagramm

Das Suchen eines geeigneten Repositoriums findet auf der graphischen Benutzeroberfläche, also der Webseite statt. Das Aktualisieren des Triplestores wird durch das Starten eines entsprechenden Java-Programms auf der Kommandozeile ausgeführt. Für diesen Use-Case gibt es keine graphische Benutzeroberfläche. Da Java eine objektorientierte Programmiersprache ist, sind die Hauptkonstrukte Methoden, Klassen und Variablen. Eine Klasse enthält in der Regel einen Konstruktor und mehrere Methoden, in denen die Programmlogik implementiert ist. Methoden können einen Rückgabewert haben und ihnen können beim Aufruf mehrere Parameter übergeben werden. Ein Vorteil der objektorientierten Programmierung ist die Kapselung und Wiederverwendung von Quellcodebestandteilen. Da ich das Open Source-Framework Sesame, aber auch die Guava-Programmbibliotheken von Google verwendet habe, wurde mir durch diese einige Implementierungsarbeit abgenommen. Programmbibliotheken können in neue Java-Programme importiert werden und bringen damit neue Klassen und Methoden mit, die der Programmierer wiederverwenden kann. Mit diesem Werkzeugkasten habe ich selbst Klassen und Methoden entworfen, die dazu dienen sollen, einen Triplestore mit RDF-Daten zu füllen. Abbildung 3

zeigt die zwei Klassen mit ihren Methoden, die ich entworfen habe und außerdem noch ein Interface und dessen Implementierung, die notwendig ist, um mit XPATH XML-Tags zu adressieren und Inhalte, die von XML-Tags umschlossen sind, auszulesen.

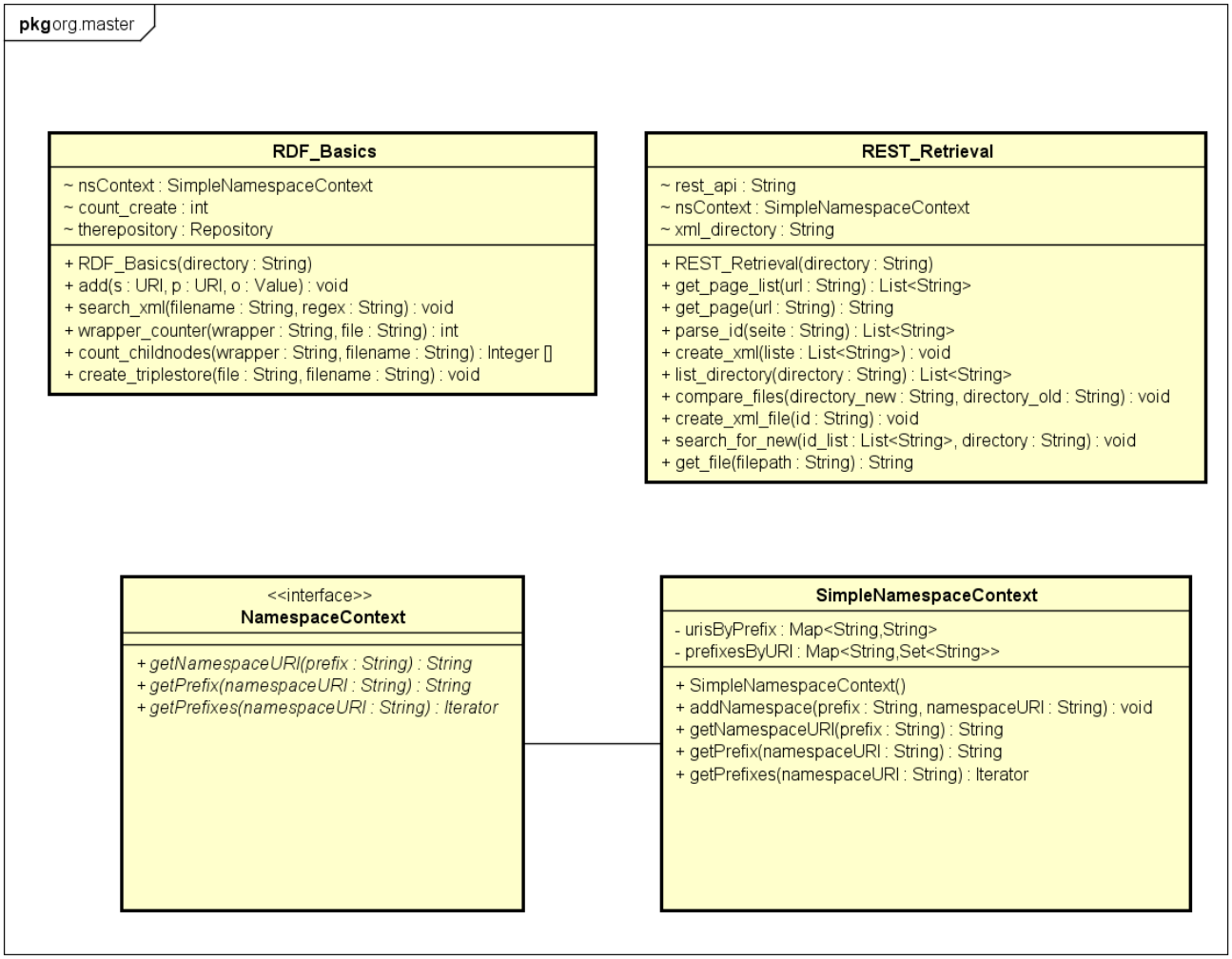

Abbildung 3: Klassendiagramme und Interface-Implementierung aus: (McLaughlin and Edelson 2006, S. 182-183)

Die Klasse „REST_Retrieval“ hat drei Instanzvariablen: rest_api, nsContext und xml_directory. Instanzvariablen sind immer für alle Methoden innerhalb der Klasse sichtbar und können von ihnen genutzt werden. Kopien dieser Variablen erscheinen in jeder separaten Instanz ihrer Klasse (Niemeyer and Leuck 2013, Vgl. S. 43). Die Tilde vor den Instanzvariablen bedeutet, dass jede Klasse in demselben Package (hier: org.master) auf diese Instanzvariablen zugreifen kann (Miles and Hamilton 2006, Vgl. S. 70). Dies ist in Java der Standard, wenn die Sichtbarkeit durch kein zusätzliches Schlüsselwort verändert wird. Hinzuzufügen ist außerdem, dass allen Methoden innerhalb der Klassendiagramme das „public“-Schlüsselwort vorangeht, gekennzeichnet durch das Pluszeichen. Dies bedeutet, dass sie für alle Klassen, auch für die in anderen Packages, uneingeschränkt sichtbar sind und auf sie von jeder Klasse zugegriffen werden kann (Miles and Hamilton 2006, Vgl. S. 68). Die Methoden, die ich in „REST_Retrieval“ implementiert habe und die ich nutze, um das Füllen des Triplestores vorzubereiten, haben folgende Funktionen: „get_page_list“ und „get_page“ haben als Parameter eine URL und geben als Rückgabewert die Daten, die sich hinter der URL verbergen, zurück - als String bei „get_page“ und als Arrayliste vom Typ

String bei „get_page_list“. „parse_id“ nimmt als Parameter einen String entgegen, durchsucht diesen String mit Hilfe eines regulären Ausdrucks und gibt als Rückgabewert alle re3data-IDs in einer Arrayliste vom Typ String zurück. Die Reihenfolge der Methodenaufrufe sollte so sein, dass zunächst „get_page“ die Übersichtsseite der re3data-REST-API lädt, um diese dann „parse_id“ zu übergeben, damit diese Methode den eindeutigen re3data-Identifier auslesen kann. Der Rückgabewert von „parse_id“ wird „create_xml“ als Parameter übergeben. „create_xml“ arbeitet dann die Arrayliste vom Typ String ab und erzeugt für jede re3data-ID eine XML-Datei in einem vorgegebenen Verzeichnis auf der Festplatte. Als nächster Schritt wird die „list_directory“-Methode aufgerufen, um in einer Arrayliste vom Typ String die Pfadangaben jeder einzelnen XML-Datei zu speichern. Diese Arrayliste wird dann zeilenweise durchlaufen. Jede Zeile enthält eine Pfadangabe für eine Datei, deren Daten mit der „get_file“-Methode in den Arbeitsspeicher geladen werden, um dann von der Methode „create_triplestore“ in RDF-Daten umgewandelt und in einem Triplestore auf der Festplatte gespeichert zu werden. Das Aktivitätsdiagramm für diesen Ablauf zeigt Abbildung 4.

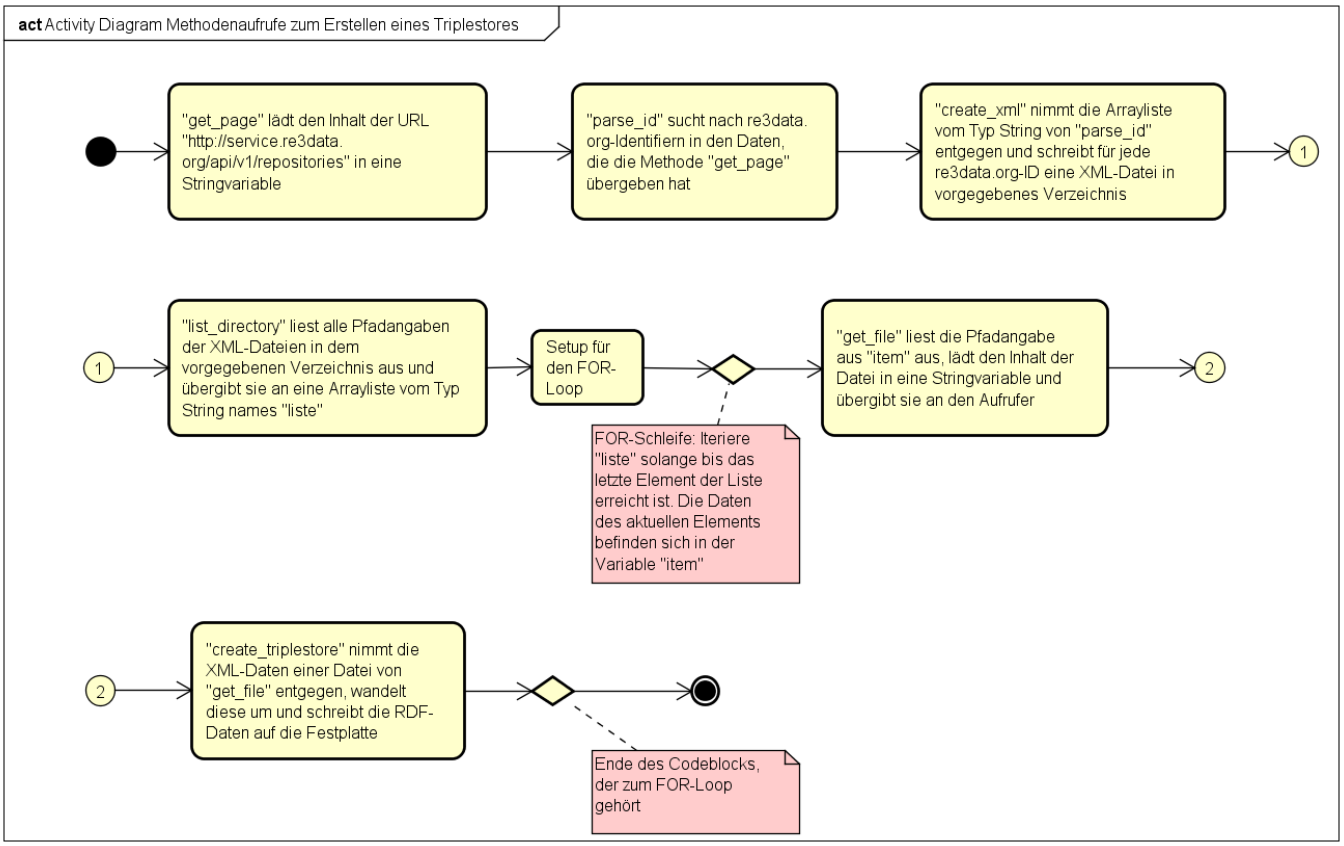

Abbildung 4: Methodenaufrufe zum Erstellen eines Triplestores

Stellt die „REST_Retrieval“-Klasse die Grundfunktionen bereit, um Daten zu lesen und zu verarbeiten, ist die Aufgabe der „RDF_Basics“-Klasse, wie der Name es schon vermuten lässt, die RDF-Daten zu generieren und in einem Triplestore abzulegen. Die Methode „search_xml“ hat dabei aber keine Bedeutung und gehört zu einem früheren Entwicklungsstand. Die „add“-Methode dagegen ist essentiell und schreibt die Triplestatements, die ihr als Parameter übergeben werden. Von ebenso großer Wichtigkeit ist die Instanz der „Repository“-Klasse, die den Triplestore zum Befüllen initialisiert und am Ende auch wieder schließt. Kommen wir nun zum Erstellen eines Triplestores in Abbildung 5:

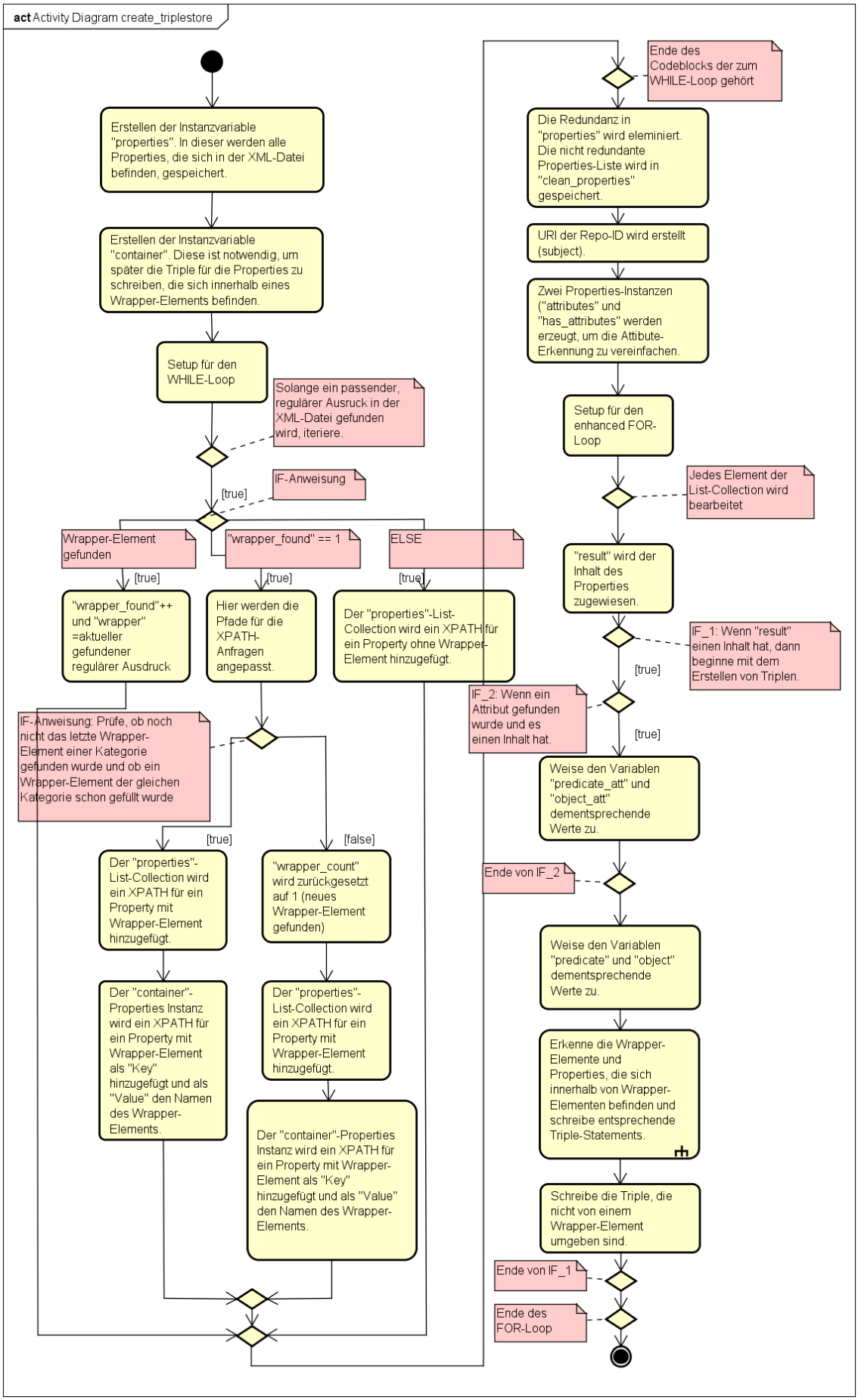

Abbildung 5: Erstellen eines Triplestores

Die Methode „create_triplestore“ empfängt als Parameter sowohl den Inhalt der zu bearbeitenden XML-Datei als String als auch deren Dateipfad. Dies ist notwendig, um sowohl den Inhalt der Datei mit regulären Ausdrücken auswerten zu können, indem eine Instanz der „Matcher“-Klasse mit dem Parameter vom Typ String erzeugt wird, als auch um eine Instanz der Klasse „InputSource“ mit dem Pfadnamen der XML-Datei zu erstellen, die dann mit XPATH-Ausdrücken ebenfalls ausgewertet werden kann. Die vorletzte Aktivität in der Diagrammabbildung 5 verweist auf einen Codeblock, den ich aus Platzgründen in ein weiteres Aktivitätsdiagramm ausgelagert habe, siehe hierzu Abbildung 6:

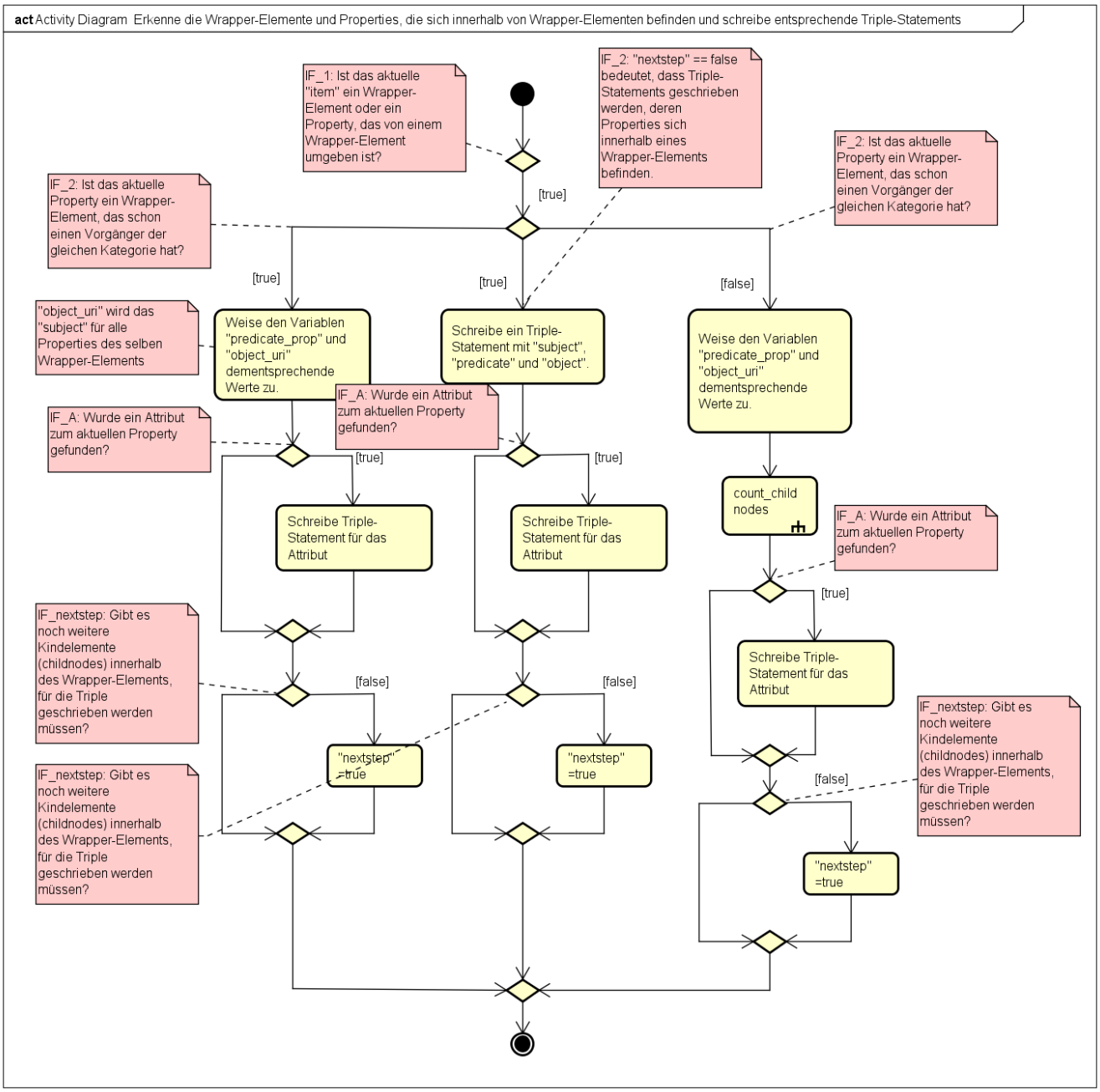

Abbildung 6: Erkenne die Wrapper-Elemente und Properties, die sich innerhalb von Wrapper-Elementen befinden und schreibe entsprechende Triple-Statements

Innerhalb von „create_triplestore“ wird an einer Stelle außerdem noch die Methode „wrapper_counter“ eingesetzt. Diese Methode nimmt als Parameter den Namen eines Wrapper-Elements entgegen und sucht nach diesem Wrapper-Element in dem (String-)

Inhalt der ihr übergebenen Datei. Als Rückgabewert liefert sie die Anzahl eines bestimmten Wrapper-Elements innerhalb der Datei. Der Algorithmus der Methode „count_childnodes“, die in Abbildung 6 ebenfalls aufgerufen wird, befindet sich in Abbildung 7:

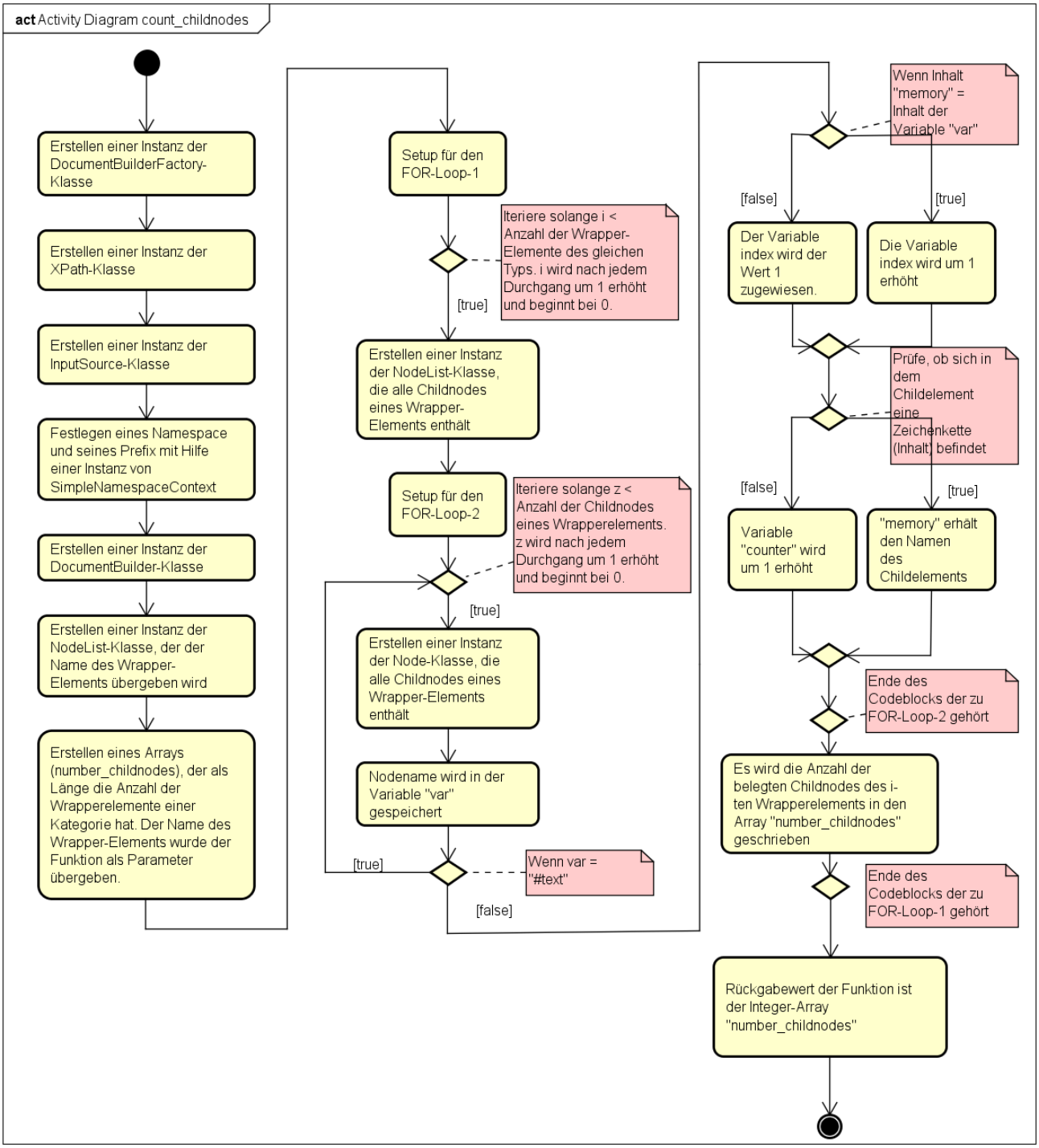

Abbildung 7: Zähle die childnodes in XML

Nachdem der Triplestore mit Hilfe des von mir vorgestellten Java-Programms gefüllt wurde, kann nun die Abfrage dieser Datenbank beginnen. Die Abfrage kann über die im Literaturbericht erwähnte REST-Schnittstelle erfolgen. Es reicht in diesem Fall aus, SPARQL-Abfragen per HTTP-GET-Request an die URL „http://localhost:8080“ zu schicken. Damit der Nutzer dieses Webservice das nicht manuell machen muss, habe ich die Webseite entwickelt, die ihm bekannte Auswahldialoge und -Möglichkeiten bietet, um nach geeigneten

Forschungsdatenrepositorien zu suchen. Bevor ich im Anschluss die Details des Algorithmus präsentiere, den ich in Javascript geschrieben habe, soll nochmal in Abbildung 8 ein Aktivitätsdiagramm die eben beschriebene Abfrage verdeutlichen:

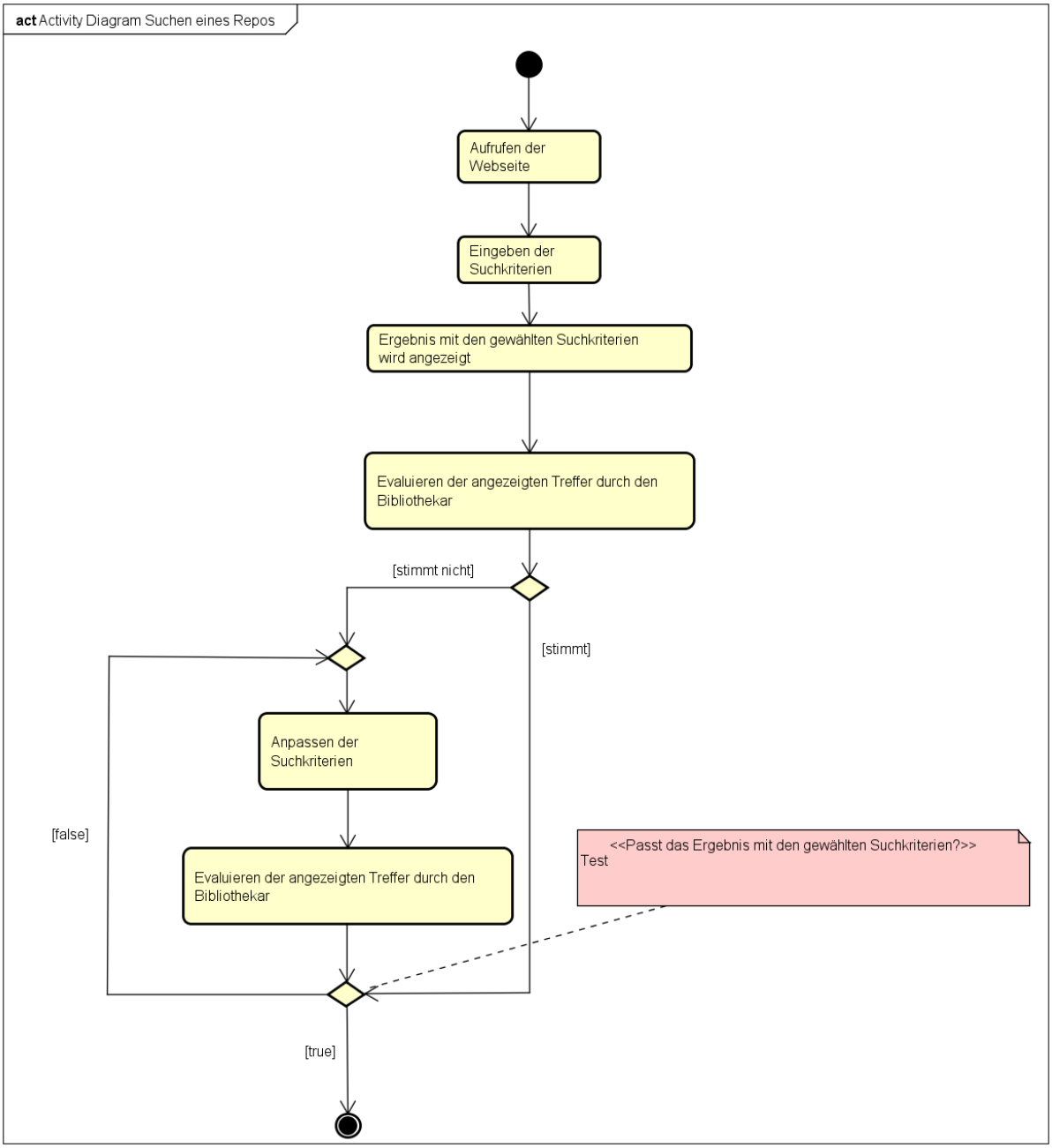

Abbildung 8: Suchen eines Repositoriums

Der Prototyp meiner Webseite läuft, wie im Literaturbericht beschrieben, auf einem Tomcat-Server unter der URL http://localhost/r3d. Wenn ein Nutzer diese Seite mit einem Webbrowser aufruft, wird der HTML- und CSS-Code interpretiert und der Javascript-Code ausgeführt. Die Implementierung der Webseitenlogik in Javascript erfolgt prozedural. Das bedeutet, dass ich für unterschiedliche Aufgaben Funktionen bzw. Methoden geschrieben habe, die in einer bestimmten Reihenfolge ausgeführt werden, wenn die Seite geladen wird. In der folgenden Abbildung 9 sieht man die Ausführung verschiedener Javascript-Funktionen, die ich implementiert habe. Dabei ist noch wichtig zu wissen, dass mit Ausnahme der Funktionen „show_repos“, „ajaxStop“ und „writehtml“ alle Funktionen

asynchron ausgeführt werden. Das heißt, dass der Aufrufer der Funktion nicht wartet, bis die Funktion beendet wird oder ein Ergebnis zurückliefert, sondern sofort nach dem Aufruf der vorhergehenden Funktion die nächste aufruft (Miles and Hamilton 2006, Vgl. S. 116).

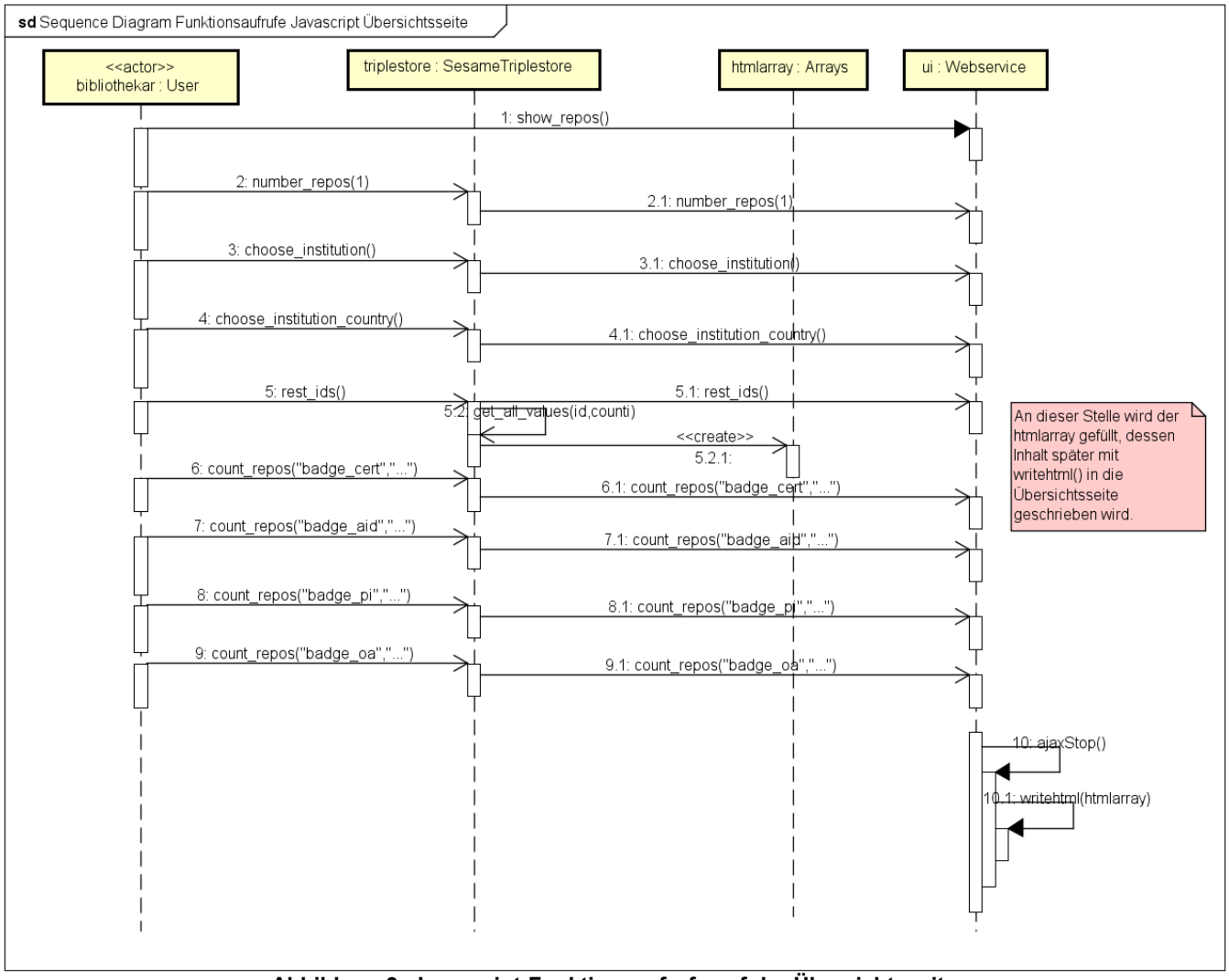

Abbildung 9: Javascript-Funktionsaufrufe auf der Übersichtsseite

Wie in Abbildung 9 zu sehen ist, werden alle Nachrichten bzw. Funktionen vom Nutzer aufgerufen, der die Webseite in seinem Browser aufruft. Die aufgerufenen Funktionen bzw. Nachrichten veranlassen ihren Empfänger etwas zu tun. Bei den Nachrichten 2, 3, 4, 5, 6, 7, 8 und 9 wird der Triplestore abgefragt, um dann das Ergebnis dieser Abfrage an die Benutzeroberfläche (ui) weiterzuleiten, die daraufhin angepasst wird. Die Ausnahmen bilden die Nachricht 5.2, die dazu führt, dass ein Array mit HTML-Code befüllt wird, und die Nachrichten 1 und 10. Die Nachricht 1 verändert nur die Benutzeroberfläche und die Nachricht 10 wartet bis alle asynchronen Funktionen beendet sind, um dann den HTML-Code in der Variable „htmlarray“ in die Übersichtsseite zu schreiben. Die Funktionen „show_repos“ und „number_repos“ implementieren die Funktionalität des Paginationsbalkens und die Funktionen „rest_ids“ und „get_all_values“ holen sich die Identifikatoren der re3data-Datensätze (rest_ids), um dann zu jeder ID alle benötigten Informationen abzufragen (get_all_values). Die Funktionen „choose_institution“ und „choose_institution_country“ befüllen die jeweiligen Dropdown-Menüs der Webseite und die übrigen Funktionen (6-9)

fragen die Anzahl der Treffer für ein bestimmtes Suchkriterium ab und schreiben die ermittelte Zahl in die blauen Badges auf der Webseite. Will nun der Nutzer auf der Seite für ein bestimmtes Repositorium mehr Informationen erhalten, kann er auf den Namen des Repositoriums auf der Übersichtsseite klicken und damit die Detailansicht öffnen, deren Sequenzdiagramm in Abbildung 10 beschrieben wird.

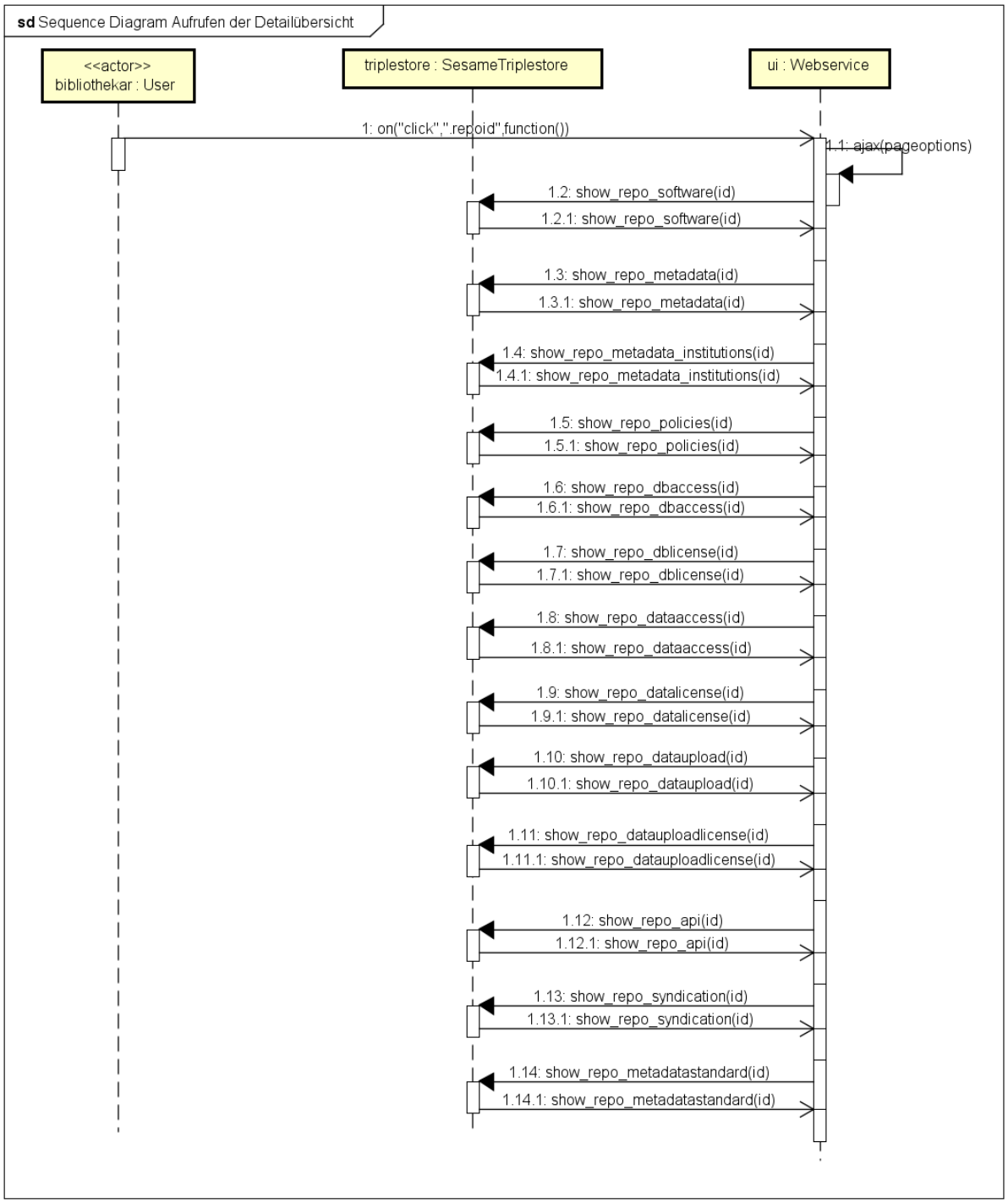

Abbildung 10: Aufrufen der Detailübersicht

Das Muster des Ladens von weiteren Detailinformationen ist bei allen Funktionen ab 1.2 gleich. Doch zunächst löst ein Klick auf ein Repositorium eine synchrone Funktion aus, die das HTML-Gerüst für die Detailinformationen lädt. Es folgen weitere synchrone Funktionen,

die immer als Erstes den Triplestore abfragen und die damit gewonnenen Informationen verwenden, um die Webseite der Reihe nach mit Detailinformationen anzureichern.
Ein weiteres wichtiges Detail der Implementierung sind die Label, die auf der Webseite erscheinen, wenn der Nutzer entweder ein Suchkriterium über ein Dropdown-Menü ausgewählt hat oder auf ein Label eines Repositoriums klickt, das sich auf der Übersichtsseite befindet. Wenn der Nutzer dieses Suchkriterium verwerfen möchte, muss er nur noch auf das entsprechende farbige Label unter den Dropdown-Menüs klicken und schon verschwindet es und das Ergebnis der Suche wird automatisch angepasst. Das Aktivitätsdiagramm für diesen „Dropdown“-Algorithmus sieht folgendermaßen aus:

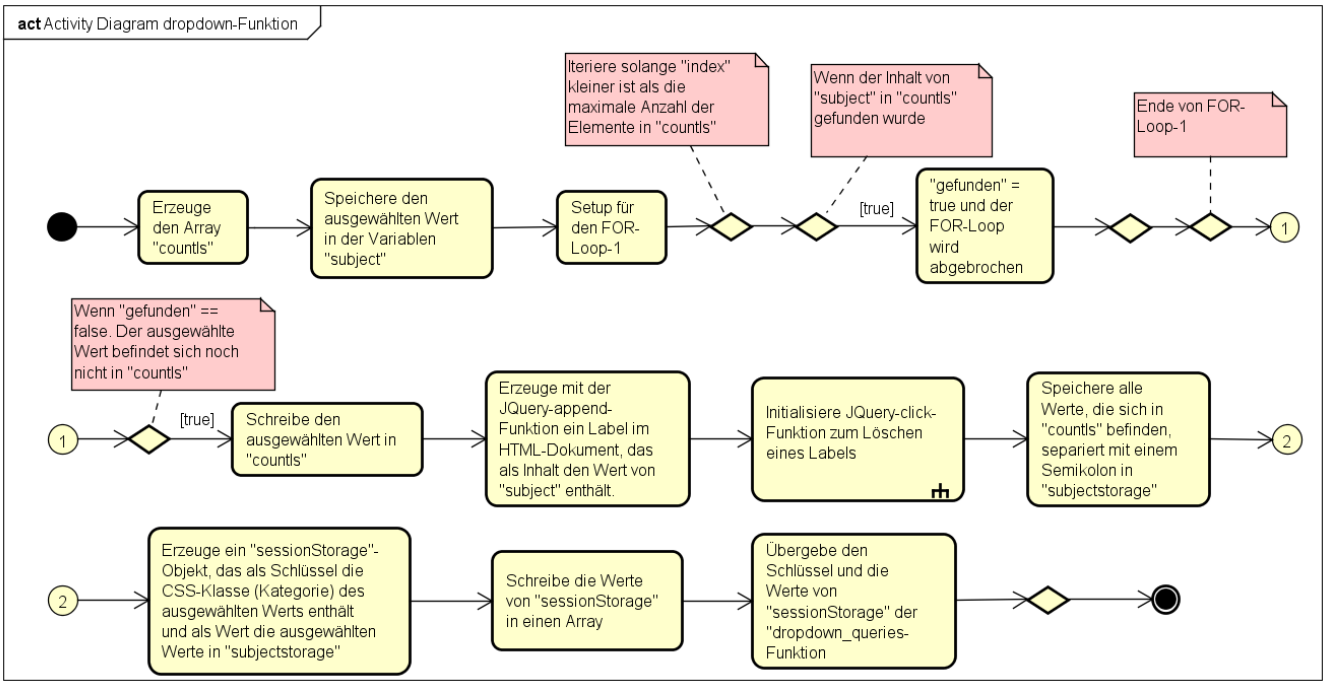

Abbildung 11: Dropdown-Funktion

Das Aktivitätsdiagramm für den Verweis auf das „Initialisieren der JQuery-click-Funktion zum Löschen eines Labels“ ist:

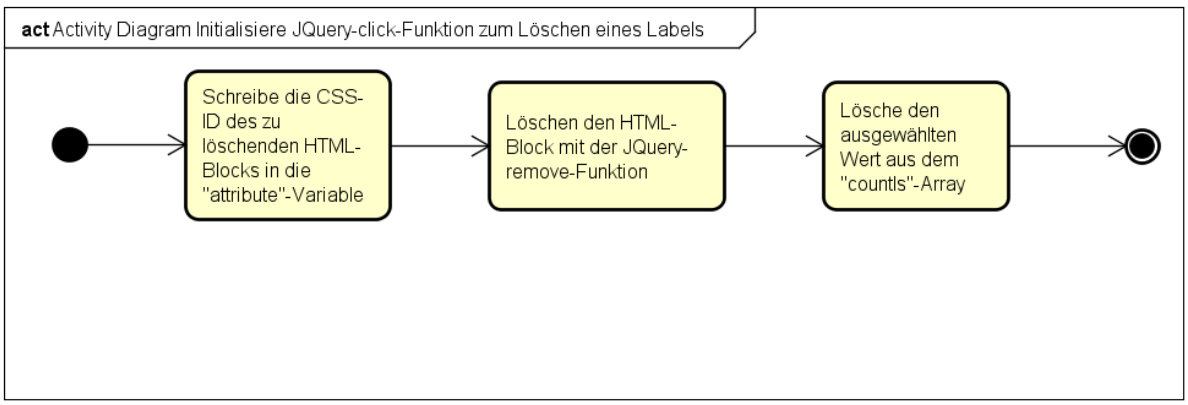

Abbildung 12: Initialisiere JQuery-click-Funktion zum Löschen eines Labels

Nachdem ich die Hauptfunktionen und Funktionalitäten meiner Javascript/JQuery-Implementierung mit Hilfe der UML-Diagramme dargestellt habe, möchte ich noch einen Blick auf die Abfragesprache SPARQL und ihre Besonderheiten werfen, die mir während der Implementierung aufgefallen sind: In der aktuellen SPARQL-Version gibt es kein Schlüsselwort für ein logisches UND. Stattdessen gibt es aber das Schlüsselwort „UNION“,

das für ein logisches ODER genutzt werden kann. Das Fehlen eines Schlüsselworts für das logische UND ist auf den ersten Blick nicht weiter wichtig, da es implizit vorhanden ist, wenn der Entwickler bspw. eine Variable für den Subjekt-Teil einer Anfrage definiert, für den Prädikat-Teil das gleiche Prädikat verwendet und im Objekt-Teil den gesuchten Literal. Der folgende Beispielcode soll dies veranschaulichen:

```
PREFIX prop: <http://re3data.org/properties/>
SELECT ?id
where {
?site prop:re3data.orgIdentifier ?id.
?site prop:subject „21 Biology".
?site prop:subject „206 Neurosciences".
}
```

Quellcodebeispiel 1: Logisches UND

Das Ergebnis dieses SPARQL-Querys wären alle Repositorien (identifiziert durch die ID), die sowohl Biologie als auch die Neurowissenschaften als ihr Fachgebiet haben. Diese logische UND-Verknüpfung funktioniert durch die Metadatenstrukturen, die auch im re3data-Fall Wrapper-Elemente umfassen können, nicht mehr wie gewünscht. Der folgende Code soll dies verdeutlichen:

```
PREFIX prop: <http://re3data.org/properties/>
PREFIX api: <http://re3data.org/properties/api/>
SELECT ?id
where {
?site prop:re3data.orgIdentifier ?id.
?site <http://re3data.org/properties/api> ?apiwrapper.
?apiwrapper api:apiType „OAI-PMH".
?apiwrapper api:apiType „REST".
}
```

Quellcodebeispiel 2: Logisches UND funktioniert liefert nicht das gewünschte Ergebnis

Das Problem ist hier, dass es nur einen Platzhalter für das Wrapper-Element gibt, nämlich „`?apiwrapper`". Wird dieser Platzhalter für das Wrapper-Element ein zweites Mal verwendet, wie im Beispiel 2, wird kein Endergebnis zurückgeliefert, weil es in jedem Wrapper-Element nur einen „apiType" laut re3data-Metadatenschema geben kann. Um dieses Problem zu lösen und auch nach Repositorien suchen zu können, die sowohl die Schnittstelle „OAI-PMH" als auch „REST" vorweisen, verwende ich das SPARQL-Schlüsselwort „`VALUES`". VALUES führt zwar auch nicht direkt zu einer logischen UND-Verknüpfung, der Entwickler kann aber für eine Variable mehrere Werte angeben, die miteinander ODER-verknüpft sind. Wird dann die Variable, die vor dem VALUES-Schlüsselwort steht, auch in die SELECT-Anweisung geschrieben, wird für diese Variable eine Spalte in der Ergebnisausgabe verwendet, in der die Werte stehen, die in der VALUES-Anweisung im WHERE-Teil eingetragen wurden. Quellcodebeispiel 3 zeigt diese Veränderung hin zu einer indirekten logischen UND-Verknüpfung:

```
PREFIX prop: <http://re3data.org/properties/>
PREFIX api: <http://re3data.org/properties/api/>
SELECT ?site ?name WHERE
{ ?site prop:repositoryName ?name.
{ SELECT ?site WHERE
{ SELECT DISTINCT ?site ?apitype
WHERE {
?site prop:re3data.orgIdentifier ?id.
?site <http://re3data.org/properties/api> ?apiwrapper.
?apiwrapper api:apiType ?apitype.
VALUES ?apitype { „OAI-PMH“ „REST“ }
}}
GROUP BY ?site HAVING (COUNT(?site) = 2) }} ORDER BY ASC(?name)
```

Quellcodebeispiel 3: Indirektes logisches UND mit SPARQL

Um die Logik hinter dieser SPARQL-Anfrage zu verdeutlichen, möchte ich auf die folgende Tabelle verweisen:

?site	?apitype
siteA	OAI-PMH
siteB	OAI-PMH
siteC	OAI-PMH
siteA	REST
siteC	REST
siteD	REST

Tabelle 1: Veranschaulichung der SPARQL-Logik zum indirekten logischen UND

Die grün hinterlegten Zeilen bedeuten, dass der Treffer zählt, weil die ID zu einer Seite sowohl bei dem Wert „OAI-PMH“ als auch beim Wert „REST“ auftaucht. Diese Logik kann der Entwickler beliebig fortsetzen. Würde bspw. ein weiteres „VALUES“-Schlüsselwort mit den Werten „Databases“, „Images“ und „Raw data“ eingefügt werden, müsste eine Seiten-ID sechsmal in der Spalte „?site“ auftauchen, wenn alle fünf Werte zu ihr gehören sollen. Außerdem müsste der „HAVING“-Teil der Suchanfrage angepasst werden: „`HAVING (COUNT(?site) = 6)`“. Mit diesem kleinen Exkurs wollte ich ein, aus meiner Sicht, wichtiges Detail meiner Implementierung darstellen. Bevor ich zur Darstellung anderer Technologien zur Implementierung eines Webservice im nächsten Kapitel komme, möchte ich für die Technologien, die ich in diesem Abschnitt präsentiert und eingesetzt habe, einen Blick auf ihre Wartbarkeit und Pflege und ihre Unterstützung durch die Web-Community werfen.

2.2 Ranking und Aspekte der eingesetzten Technologien

Die erste Annahme in diesem Zusammenhang ist, dass, wenn eine Programmiersprache ein hohes Ranking in einem Index hat, sie auch von vielen Entwicklern verwendet wird. Der PYPL-Index (PYPL-Index 2016) untersucht bspw. wie oft Programmiersprachen-Tutorials auf

Google gesucht werden. Daraus wird dann ein Ranking erstellt und auf die Popularität geschlossen. Die Programmiersprache Java ist im Januar 2016 auf Platz 1. Java führt außerdem noch den TIOBE-Index (BV 2016) vom Januar 2016 an und ist auf dem zweiten Platz beim RedMonk-Ranking vom Juni 2015 (RedMonk 2015). Beim TIOBE-Index werden neben den Suchanfragen auf Google, die die Form "<language> programming" (Neumann 2016) haben, noch weitere 24 Suchmaschinen ausgewertet. Die Form der Suchanfrage bleibt dabei gleich. Beim RedMonk-Ranking werden dagegen die Community und Quellcode-Seiten Stack Overflow und GitHub ausgewertet. Die Analyse von Stack Overflow bezieht sich auf die Diskussionen über eine Sprache, bspw. Problemlösungen, und bei GitHub auf die tatsächliche Nutzung durch eine Messung der Quellcodezeilen in einer Sprache. Die Analyse der Stack Overflow Diskussionen zeigt, ob ein Entwickler auf die Erfahrung und Problemlösungen der Web Community zurückgreifen kann. Die Analyse der Suchanfragen beim TIOBE-Index und der Quellcodezeilen auf GitHub kann Auskunft darüber geben, ob z. B. eine Firma auf eine große Menge von potenziellen Programmierern zurückgreifen kann, wenn sie sich für eine Sprache entscheidet. Auch für Javascript kann eine Firma zu solchen Schlussfolgerungen kommen: Sie belegt Platz 1 im RedMonk-Rating; die beiden anderen Rankings sehen sie zumindest noch in den Top Ten. Ein Vorteil von Javascript ist auch die große Anzahl von Frameworks, wie bspw. JQuery, um in kurzer Zeit Applikationen schreiben zu können. Allerdings werden die Programmbibliotheken oft aktualisiert, was den Wartungsaufwand erhöht (Vgl. Reeves 2015). „Die Herausgeber [des TIOBE-Index] (…) sehen für dieses Jahr außerdem positive Entwicklungen bei PHP, das Ende des vergangenen Jahres nach vielen Jahren der Entwicklung ein neues Major-Release verzeichnen konnte.“ (Neumann 2016) Im RedMonk-Index und im PYPL-Index befindet sich PHP auf Platz 3. Ein Vorteil von PHP sind, ebenso wie bei Javascript, seine Frameworks für das „Rapid Application Development“ (Vgl. Reeves 2015). Ein PHP-Framework, mit dem ich mich im nächsten Kapitel näher beschäftigen werde, ist Symfony, mit dem der Webentwickler einen Webservice mit Datenbankanbindung implementieren kann. Der re3data.org-Webservice verwendet als einen Bestandteil Symfony, was dazu führt, dass ich eine alternative, theoretische Implementation mit Symfony auf den folgenden Seiten darstellen möchte.

2.3 Alternative, theoretische Implementierung eines Webservice

Bevor ich mit unterschiedlichen Diagrammen und einem Quellcodebeispiel eine weitere mögliche Implementierung eines Webservice darstellen werde, möchte ich noch einen Blick auf die Software-Bestandteile werfen, die dabei zum Einsatz kommen. Dabei möchte ich die Komponenten mit der von mir entwickelten Implementierung vergleichen:

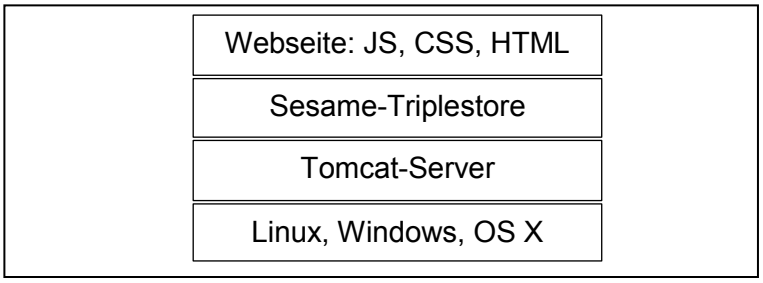

Abbildung 13: Software-Komponenten meiner Implementierung

Wie man sehen kann, kommen nur vier Komponenten zum Einsatz. Auf der untersten Ebene kann der Entwickler sich für ein Betriebssystem entscheiden, auf dem eine virtuelle Maschine der Programmiersprache Java läuft. Diese ist auch notwendig für den Betrieb des Tomcat-Servers, auf dem der Sesame-Triplestore läuft, der als eine Datenbank fungiert. Auf der obersten Ebene laufen die Skriptsprache Javascript, HTML und die Formatierungssprache CSS. Bei der Implementierung mit dem PHP-Framework Symfony werden dagegen, wie in Abbildung 14 sichtbar, z. T. andere Software-Komponenten eingesetzt:

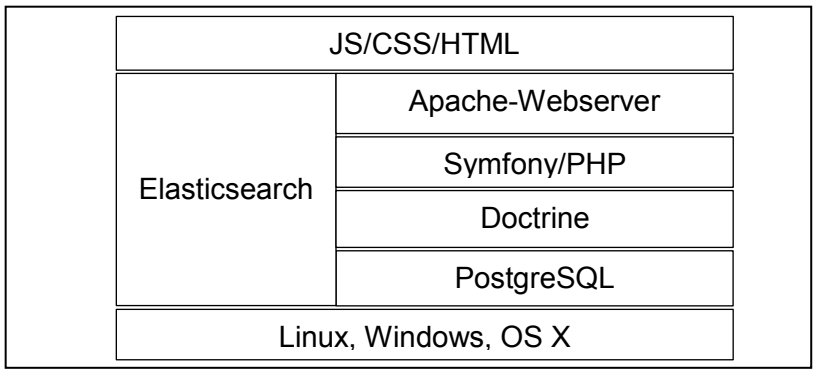

Abbildung 14: Implementierung mit Symfony und Elasticsearch

Der erste Unterschied ist die Verwendung anderer Datenbankmanagementsysteme (DBMS). PostgreSQL ist ein relationales DBMS, das seine Daten in Tabellen speichert. Zur Darstellung der Tabellen, ihrer Beziehungen untereinander und ihrer Attribute, folgen zwei Entity-Relationship-Diagramme:

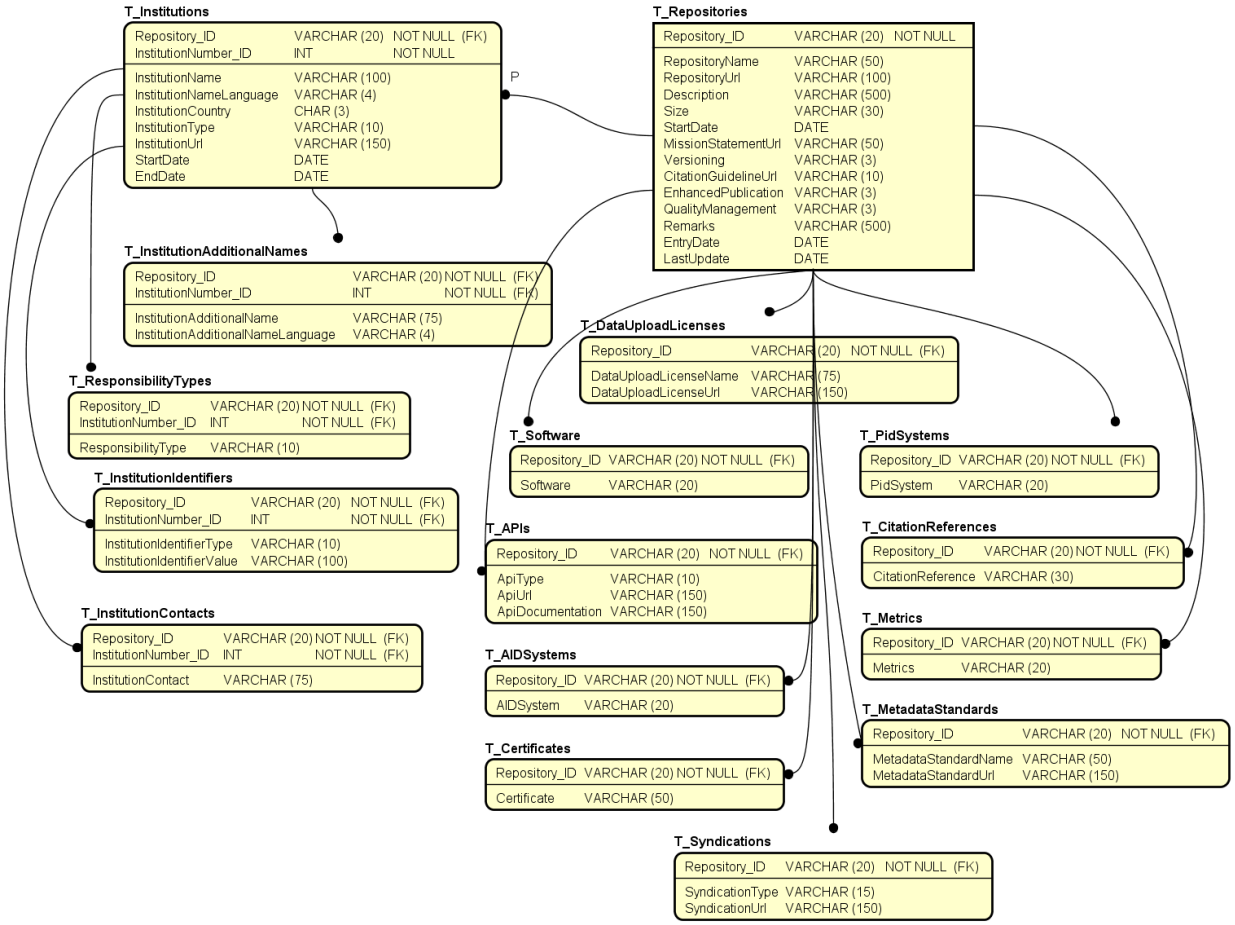

Abbildung 15: ER-Diagramm 1

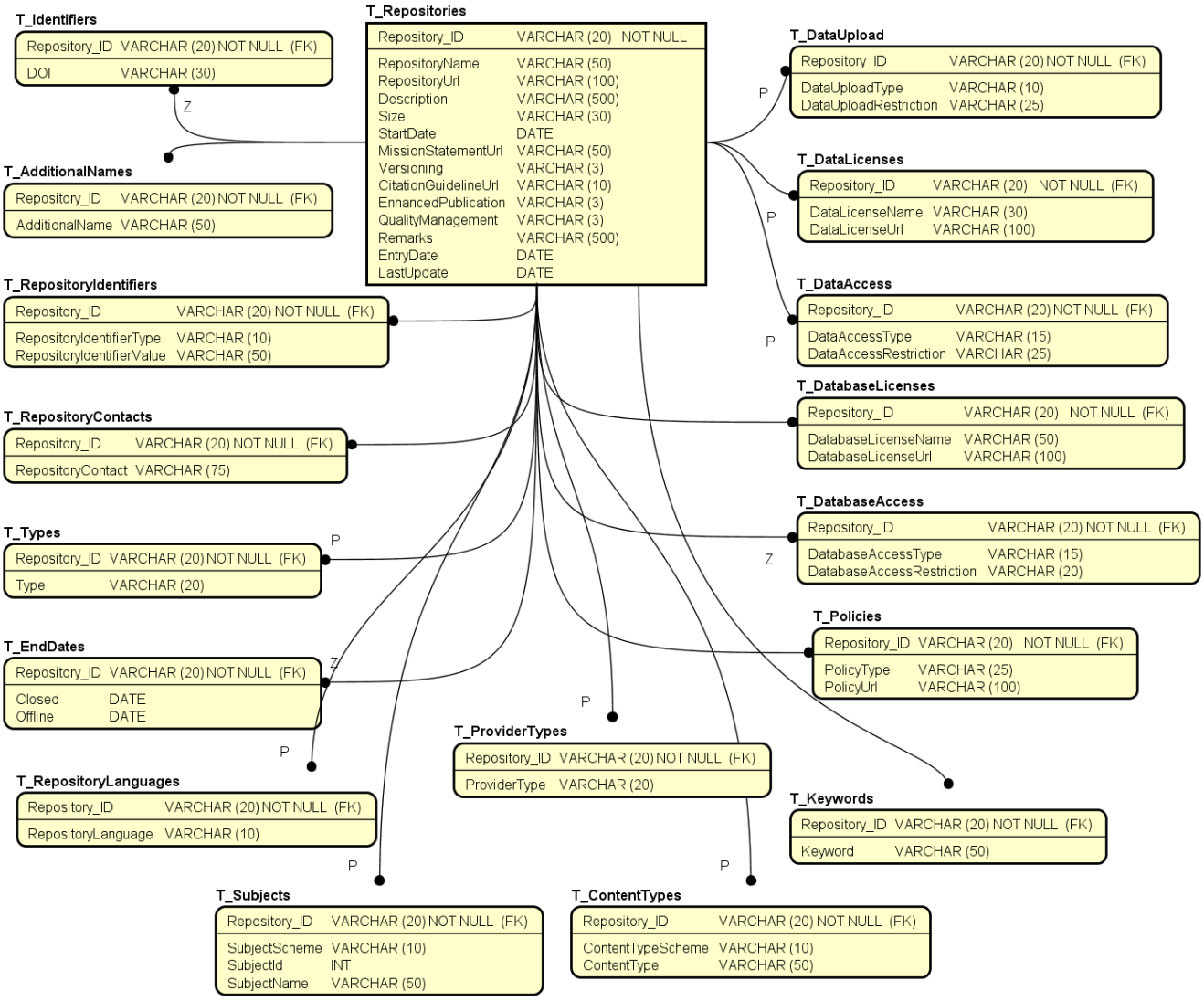

Abbildung 16: ER-Diagramm 2

Diese beiden ER-Diagramme sind nach dem re3data-Metadatenschema 3.0 modelliert und sind als ein Ganzes zu sehen, nur aus Platzgründen habe ich zwei erstellt. Die Daten des von mir gefüllten Triplestores entsprechen dagegen dem re3data-Metadatenschema 2.2, da bis jetzt die REST-API des Projekts nur XML-Dateien nach diesem Schema ausgibt und ich aus zeitlichen Gründen die Java-Programme zur Generierung des Triplestores nicht mehr anpassen kann. Wichtig zu wissen ist, dass die eckige Tabelle „T_Repositories“ den Primärschlüssel „Repository_ID“ enthält. In jeder Beziehung zu anderen Tabellen, die eine runde Form haben, ist dieser Primärschlüssel dort der Fremdschlüssel. Der Fremdschlüssel identifiziert dann in der Regel alleine alle Nichtschlüsselattribute für die mit „T_Repositories“ verknüpften Tabellen. Nur in der Tabelle „T_Institutions“ setzt sich der Primärschlüssel aus dem Fremdschlüssel von „T_Repositories“ und einem weiteren Attribut („InstitutionNumber_ID“) zusammen (siehe Abb. 15). Für die von „T_Institutions“ wiederum abhängigen Tabellen „T_InstitutionAdditionalNames“, „T_ResponsibilityTypes“, „T_InstitutionIdentifiers“ und „T_InstitutionContacts“ ist in diese Attributkombination dort der Fremdschlüssel von „T_Institutions“ und identifiziert in den vier erwähnten Tabellen auch alle Nichtschlüsselattribute. Diese Tabellen sind auch in der zweiten Normalform, da sie in der

ersten Normalform sind und die Nichtschlüsselattribute von ihrem zusammengesetzten Schlüssel voll funktional abhängig sind. Neben diesen erwähnten Tabellen sind auch alle weiteren Tabellen in der dritten Normalform, da sie in der zweiten Normalform sind und kein Nichtschlüsselattribut von seinem Schlüssel transitiv abhängig ist. Die Beziehungen zwischen der Tabelle „T_Repositories“ und den mit ihr verknüpften Tabellen können aus drei unterschiedlichen Beziehungen bestehen. Es gilt aber immer nur eine 1 zu 1-, eine 0 zu N- oder eine 1 zu N-Beziehung. In den beiden ER-Diagrammen wird eine 1 zu 1-Beziehung durch ein „Z“ gekennzeichnet, eine 0 zu N-Beziehung besitzt keinen Buchstaben und eine 1 zu N-Beziehung wird durch ein „P“ ausgewiesen. Nach Abbildung 14 setzt auf der relationalen PostgreSQL-Datenbank der Doctrine Object Relational Mapper (ORM) auf. Doctrine beinhaltet auf der unteren Ebene einen Database Abstraction Layer (DBAL), der unterschiedliche relationale DBMS ansprechen kann. Die DBAL kann den gleichen SQL-Query auf verschiedenen DBMS ausführen, indem sie ihn intern umschreibt und anpasst. Der darauf aufsetzende ORM erlaubt den Zugriff und das Verwalten relationaler Datenbanktabellen und -reihen durch eine objektorientierte API (Dunglas 2013, Vgl. S. 5). Diese API kann in Symfony mit PHP angesprochen werden. Im Folgenden möchte ich eine Beispielkonfiguration für den Doctrine ORM anhand eines Quellcodebeispiels zeigen:

```php
<?php  namespace Blog\Entity;
use Doctrine\ORM\Mapping\Entity;
use Doctrine\ORM\Mapping\Column;
use Doctrine\ORM\Mapping\ManyToOne;
/**
 * T_RepositoryContacts entity
 * @Entity
 */
class T_RepositoryContacts
{
    /**
     * @var string
     *
     * @Column(type="string")
     */
    protected $RepositoryContact;
    /**
     * @var T_Repositories
     *
 *@ManyToOne(targetEntity="T_Repositories",inversedBy="contacts")
     */
    protected $repository;
/* Es folgen noch Getter- und Setter-Methoden */
}
```

Quellcodebeispiel 4: Many-To-One

```php
<?php
namespace Blog\Entity;
use Doctrine\ORM\Mapping\Entity;
use Doctrine\ORM\Mapping\Column;
use Doctrine\ORM\Mapping\OneToMany;
use Doctrine\Common\Collections\ArrayCollection;

/**
 * T_Repositories entity
 *
 * @Entity
 *
 */
class T_Repositories
{
    /**
     * @var string
     *
     * @Column(type="string")
     */
    protected $RepositoryName;
    /**
     * @var string
     *
     * @Column(type="text")
     */
    protected $RepositoryUrl;
    /**
        * Es folgen weitere Attribute der T_Repositories-Tabelle.
        *
        */
    /**
     * @var T_RepositoryContacts[]
     *
     * @OneToMany(targetEntity="T_RepositoryContacts", mappedBy="repository")
     */
    protected $contacts;

    /**
     * Initializes collections
     */
    public function __construct()
    {
        $this->contacts = new ArrayCollection();
    }
    /**
        *
        * Es folgenden Getter und Setter-Methoden.
        */
}
```

Quellcodebeispiel 5: One-To-Many

Diese beiden Quellcodebeispiele (4 und 5) demonstrieren, wie ein Entwickler eine

bidirektionale 1 zu N-Beziehung erstellen kann. Das Quellcodebeispiel habe ich von der Doctrine-Dokumentationsseite (Doctrine 2016) und dem Buch von Kévin Dunglas (Dunglas 2013, Vgl. S. 41-43) entnommen und gemäß dem ER-Diagramm angepasst. Es wird ersichtlich, dass die Tabellen des ER-Diagramms als Klassen der objektorientierten Programmiersprache PHP beschrieben werden. Die Attribute der Tabellen werden zu den Attributen der Klassen. Die Beziehungen zwischen den Tabellen werden in dem von mir ausgesuchten Beispiel mit dem „@OneToMany“- und dem „@ManyToOne“-Schlüsselwort ausgedrückt. Bei diesem bidirektionalen Mapping befindet sich das Schlüsselwort „mappedBy“ auf der „OneToMany“-Seite und das Schlüsselwort „inversedBy“ auf der „ManyToOne“-Seite (Vgl. Doctrine 2016). Wie im ER-Diagramm ersichtlich, gibt es aber nicht nur 0 zu N- bzw. 1 zu N-Beziehungen, sondern auch 1 zu 1-Beziehungen. Möchte der Entwickler eine solche Beziehung mit Doctrine modellieren, kann er das Schlüsselwort "@OneToOne" verwenden. Bei einem bidirektionalen Mapping müssen wie bei "@OneToMany" und "@ManyToOne" auch wieder die Schlüsselwörter "mappedBy" und "inversedBy" verwendet werden. Der Fremdschlüssel befindet sich dabei immer auf der Seite, auf der das "inversedBy"-Schlüsselwort verwendet wird. Der Vollständigkeit halber soll hier noch erwähnt sein, dass es auch ein "@ManyToMany"-Schlüsselwort gibt, um N zu M-Beziehungen abzubilden. Die Unterscheidung zwischen unidirektionalen und bidirektionalen Beziehungen beeinflusst nicht das DBMS. Diese Unterscheidung ist nur für den ORM wichtig. Bei bidirektionalen Beziehungen ist es möglich, dass aus beiden Klassen auf die jeweils andere zugegriffen werden kann. Bei unidirektionalen Beziehungen ist der Zugriff nur in eine Richtung möglich. Wenn auf diese Weise für jede Datenbanktabelle eine Klasse in PHP definiert wurde und Getter- und Setter-Methoden in dieser Klasse definiert sind, ist es möglich mit Doctrine Objekte aus dem relationalen DBMS zu ziehen und als PHP-Objekt auf diese zuzugreifen. Der Entwickler kann dann eine Reihe in einer Tabelle in einem PHP-Objekt speichern und manipulieren. Wurde dieses PHP-Objekt bspw. mit einer Setter-Methode verändert, kann mit Hilfe von Doctrine das vollständige Objekt wieder in der relationalen Datenbank abgelegt werden. Dabei spielt es keine Rolle, welches konkrete relationale DBMS verwendet. Doctrine abstrahiert die Zeilen der Tabellen zu PHP-Objekten, auf die der Entwickler mit PHP-Funktionen, die Doctrine mitliefert, zugreifen kann. Mit Doctrine ist es möglich in abstrahierter Form, nämlich in Form von Objekten, auf die Daten eines Datenbankmanagementsystems zuzugreifen und diese zu verändern. Aus der Sicht des Model-View-Controller-Architekturmusters gehört Doctrine zur Komponente Modell. „Das Model umfasst die Kernfunktionalität und kapselt die Verarbeitung und die Daten des Systems. Der Grundgedanke der MVC-Architektur ist die Zerlegung einer interaktiven Anwendung in die Komponenten Model, View und Controller. Dabei ist jeder View stets ein Controller zugeordnet. Ein Controller kann aber mehrere Views ‚bedienen‘.“ (Goll 2014, S. 378) Bei der Softwareentwicklung mit Symfony ist es ratsam, diese drei Komponenten explizit zu definieren und damit zu implementieren. Die Anwendung dieses Architekturmusters in Symfony führt zu den Grundlagen des Internets: Eine Anfrage erreicht einen Server im Netz, dieser verarbeitet die Anfrage und schickt eine Antwort an den Ersteller der Anfrage. Bei Symfony werden eingehende Anfragen von der Routing-

Konfiguration des Webservice interpretiert, indem bspw. die URL interpretiert wird, mit der der Anfragende den Webservice aufruft. Ist die URL des Anfragenden mit einer URL identisch, die in der Routing-Konfiguration hinterlegt wurde, wird die Anfrage an die entsprechende Controller-Funktion weitergeleitet, die abschließend eine Antwort generiert und an der Aufrufer zurückschickt, indem bspw. die View, also der darzustellende Inhalt der Webseite verändert wird. Im Controller wird die Anfrage interpretiert und eine Antwort generiert (SensioLabs 2015, Vgl. S. 11). Ein Beispiel für eine Implementierung eines Controllers, geschrieben in PHP, soll folgen. Zunächst möchte ich ein Klassendiagramm

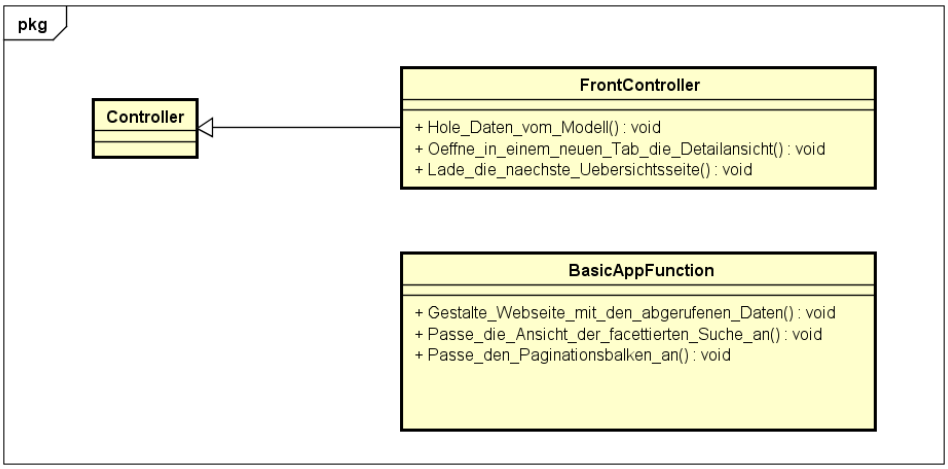

Abbildung 17: Klassendiagramm für Symfony

(Abbildung 17) zeigen, das als Vorlage dienen kann, gefolgt von einem entsprechenden Sequenzdiagramm (Abbildung 18).

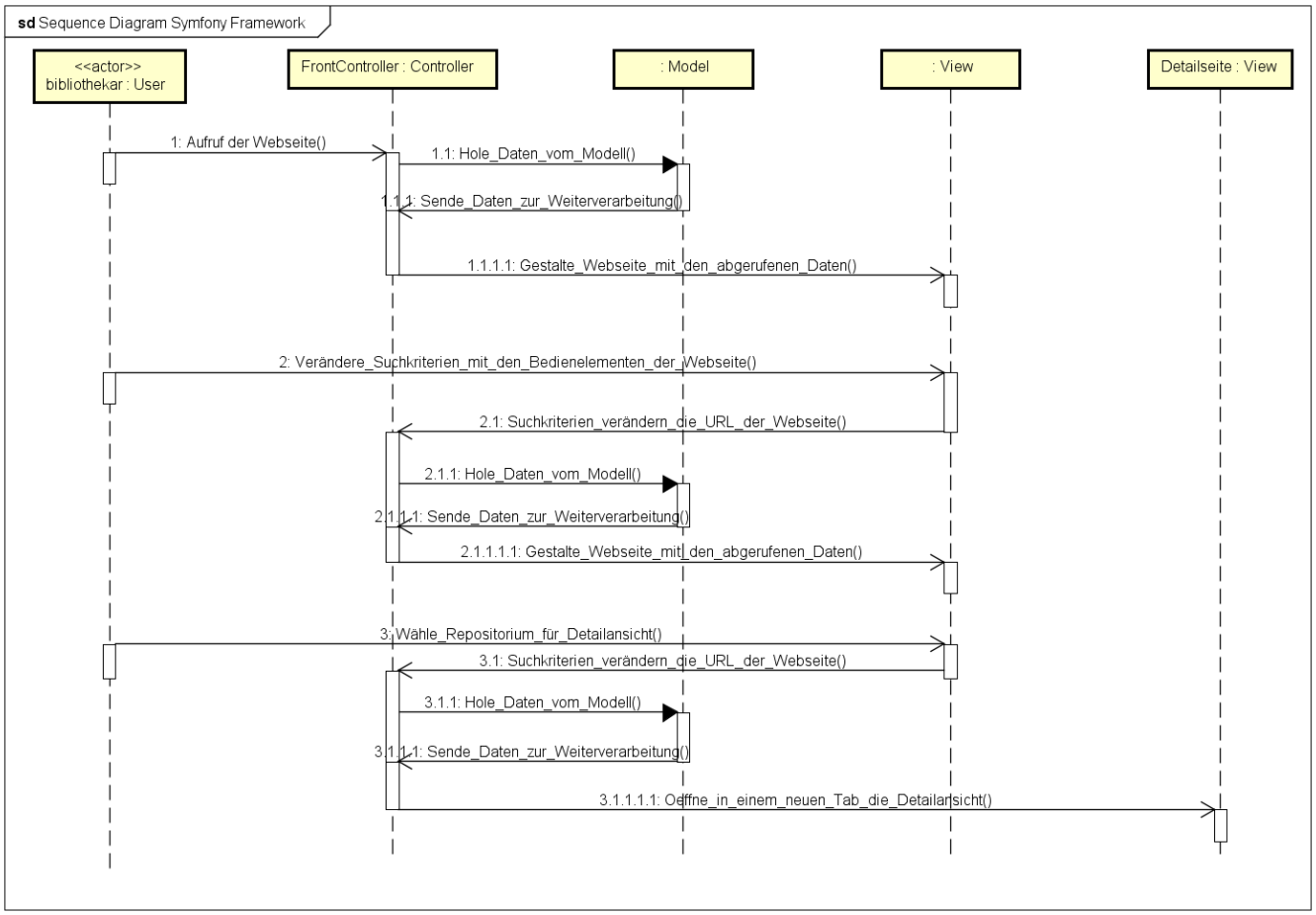

Abbildung 18: Sequenzdiagramm für Symfony

```php
<?php
namespace AppBundle\Controller;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
$basics = new BasicAppFunction();
class FrontController extends Controller
{
    /**
    * @Route("/r3d/{criteria}", defaults={"criteria" = ""})
    */
    public function Hole_Daten_vom_Modell($criteria)
    {
        $basics->Gestalte_Webseite_mit_den_abgerufenen_Daten();
        $basics->Passe_die_Ansicht_der_facettierten_Suche_an();
        //....
    }
    /**
    * @Route("/r3d/{repoid}", defaults={"repoid" = ""})
    */
    public function Oeffne_in_einem_neuen_Tab_die_Detailansicht($repoid)
    {
        //....
    }
    /**
    * @Route("/r3d/{page}", defaults={"page": 1}, requirements={
    * "page": "\d+"
    * })
    */
    public function Lade_die_naechste_Uebersichtsseite($page)
    {
        $basics->Passe_den_Paginationsbalken_an();
        //....
    }
}
class BasicAppFunction
{
    public function Gestalte_Webseite_mit_den_abgerufenen_Daten()
    {
        //....
    }
    public function Passe_die_Ansicht_der_facettierten_Suche_an()
    {
        //....
    }
    public function Passe_den_Paginationsbalken_an()
    {
        //....
    }
}
?>
```

Quellcodebeispiel 6: Controller in PHP für Symfony (Ich habe hier einen Screenshot verwendet, da Endnote einige Schlüsselwörter im Textfeld verarbeiten wollte.)

Dieser Quellcode sowie das Sequenz- und Klassendiagramm erheben keinen Anspruch auf Vollständigkeit. Sie sollen stattdessen exemplarisch aufzeigen, wie ein Entwickler nach dem MVC-Muster einen Webservice mit Symfony implementieren könnte. Wichtig zum

Verständnis des Quellcodebeispiel 6 ist das „`@Route`"-Schlüsselwort. Mit diesem werden die Routing-Informationen konfiguriert. Ruft bspw. ein Nutzer den Webservice mit der URL „`http://localhost/r3d`" auf, so würde die Funktion „`Hole_Daten_vom_Modell`" ohne Kriterien (die Variable „`criteria`" hat als Default einen leeren String) aufgerufen werden, die wiederum die Funktionen „`Gestalte_Webseite_mit_den_abgerufenen_Daten`" und „`Passe_die_Ansicht_der_facettierten_Suche_an`" aufrufen würde. Ganz ähnlich verhält es sich, wenn die URL „`http://localhost/r3d/r3d100010129`" aufgerufen würde. Dies würde dazu führen, dass die Funktion „`Oeffne_in_einem_neuen_Tab_die_Detailansicht(r3d100010129)`" mit einem Parameter aufgerufen würde. Die Funktionen innerhalb einer Controllerklasse holen sich die Daten vom Modell, formatieren die Daten für die View und schicken die formatierte Webseite als Antwort an den Aufrufer (in der Regel der Webbrowser) zurück. Wenn ein Controller die View bedienen soll, indem er HTML mit Stylesheet-Informationen ausgeben soll, kann und sollte er die Arbeit an eine Template Engine übergeben (SensioLabs 2015, Vgl. S. 71). Symfony bietet mit der Template-Sprache Twig oder alternativ mit der Nutzung von PHP-Templates eine Möglichkeit Templates zu schreiben, die in der View angezeigt werden. Fällt die Entscheidung auf Twig, erhält der Entwickler eine strikte Trennung der Präsentation von Inhalten und der Programmlogik, da Twig keine PHP-Tags verarbeiten kann. Stattdessen wird jedes Twig-Template zu einer PHP-Klasse kompiliert und zur Laufzeit gerendert (SensioLabs 2015, Vgl. S. 73). Außerdem hat der Entwickler die Möglichkeit Stylesheet- und Javascript-Dateien in Twig einzubinden (SensioLabs 2015, Vgl. S. 82). Der Inhalt von Javascript-Dateien wird in der Regel im Webbrowser des Clients ausgeführt, auch übernimmt der Webbrowser die Interpretation der HTML-Markups und Stylesheet-Informationen. Als Webserver für das Symfony-Framework bietet sich der weitverbreitete Apache-Webserver an. „Apache skaliert sehr gut, besitzt ausgezeichnete Caching-Plugins und eine gute Unterstützung für Load-Balancer und Sprayer (Maschinen, die Requests effektiv über mehrere Webserver verteilen)." (Tate and Hibbs 2007, S. 9) Der Webserver ist die Schnittstelle zwischen den darunter liegenden Technologien wie Datenbanken und Software-Frameworks. Oft liefert dieser auf eine Anfrage des Browsers nur eine View zurück oder verändert eine View durch eine entsprechende Antwort des Servers. Wie in Abbildung 14, bei der Darstellung der unterschiedlichen Schichten einer Webanwendung, ersichtlich ist, kann der Entwickler parallel zu der relationalen Datenbank, dem Symfony-Framework und dem Apache-Webserver, auch noch einen ElasticSearch-Server betreiben, der eine Volltextsuche in den Daten der Webanwendung ermöglicht. Allerdings ist auch schon eine Volltextsuche mit der PostgreSQL-Datenbank möglich. Mit Hilfe der LIKE-Abfrage kann der Entwickler „einen Datensatz anhand des Vorkommens einer bestimmten Zeichenkette aus der Datenbank" abfragen. Ferner hat der Entwickler mit der Nutzung des speziellen Datentyps „tsvector" (steht für Text Search Vector) die Möglichkeit „eine effiziente, moderne Volltextsuche direkt mit der Datenbank durchzuführen." Dieser Datentyp „ermöglicht es, diverse Transformationen auf einem Text durchzuführen, sodass dieser für die Volltextsuche indiziert werden kann." Dabei werden bspw. Stoppwörter entfernt und

Stemming angewandt. Als weitere Möglichkeit für die Verwendung einer Volltextsuche besteht die „Auslagerung der Suchlogik in ein eigenes Teilsystem“, wobei die „Synchronisation der Datensätze mit einer bestehenden Datenbank“ eine Herausforderung sein kann. ElasticSearch (ES) bietet als Teilsystem einer Webanwendung viele Vorteile, die mit einer relationalen Datenbank schwerer zu realisieren sind: Z. B. weist ElasticSearch eine sehr hohe Geschwindigkeit auf, es können „mehrere Gigabyte große Indizes effizient“ durchsucht werden. Ferner bietet ES eine Auto-Vervollständigung der eingegebenen Suchanfragen, was zu einer Fehlerreduktion der Suchanfragen des Nutzers führen kann, da er sofort auf eine fehlerhafte Eingabe hingewiesen wird. Ein weiteres Feature ist die Nutzung einer facettierten Suche: „So ist es möglich dem Nutzer mehrere Filter anzubieten, die er dann beliebig untereinander kombinieren kann, um die Ergebnisse einzuschränken.“ (Vgl. Oscity 2015) Zum Betreiben von ElasticSearch wird zur Beginn nur ein Java Development Kit (JDK), das cURL-Programm und ein Editor benötigt. „Für den ElasticSearch-Server gilt ein Zero-Configuration-Ansatz, das heißt, dass für alle möglichen Einstellungen sinnvolle Standardwerte vorgegeben sind.“ ES ist ein NoSQL-System und verfolgt einen „dokumentzentrierten Ansatz im Umgang mit Daten.“ (Vgl. Fischer 2013) „NoSQL (englisch für Not only SQL) bezeichnet Datenbanken, die einen nicht-relationalen Ansatz verfolgen (…).“ (Wikipedia 2016f) Unter einem dokumentzentrierten bzw. dokumentorientierten Ansatz versteht man „eine Datenbank, bei der Dokumente die Grundeinheit zur Speicherung der Daten bilden. (…) Während eine relationale Datenbank aus Datenbanktabellen besteht, die einem festen Datenbankschema unterliegen, enthält eine dokumentenorientierte Datenbank einzelne Dokumente.“ (Wikipedia 2016b) „Für die spätere Suche [nach diesen Dokumenten oder ihrem Inhalt] werden [die] zu indizierenden Daten als JSON-Dokumente an ElasticSearch übergeben und gespeichert.“ „Läuft ElasticSearch, lassen sich JSON-Dokumente mit einem HTTP-POST- oder HTTP-PUT-Befehl zur Indizierung übergeben.“ ES bietet dafür eine REST-Schnittstelle, über die sich jede Ressource in ES über eine REST-konforme URL ansprechen lässt. (Vgl. Fischer 2013) Ähnlich wie bei der oben erwähnten PostgreSQL-Datenbank und ihren Funktionen zur Volltextsuche werden bei ElasticSearch mit Hilfe von sogenannten Analyzern Eingangstexte in einzelne Token aufgespalten und vor dem Speichern in den Indizes gefiltert und transformiert. „Mit Analyzern lassen sich zum Beispiel Texte in einzelne Wörter zerteilen, diese in Kleinbuchstaben umwandeln und Stoppwörter (…) herausfiltern, ehe die Token gespeichert werden.“ (Vgl. Fischer 2013) Da sowohl das Einfügen von neuen Dokumenten als auch das Retrieval von Informationen in den Dokumenten in ES über eine REST-Schnittstelle erfolgt, kann der Webentwickler zum Abfragen von Informationen bspw. die JQuery-Get-Funktion (https://api.jquery.com/jquery.get/) nutzen, die dann die Retrieval-Ergebnisse als JSON-Daten vom ES-Server entgegennimmt. Die JSON-Daten könnten dann mit weiteren Javascript-Funktionen ausgewertet und dynamisch in die Webseite integriert werden.

Mit diesen Ausführungen hoffe ich ein paar Grundzüge der Implementierung mit den Technologien nach Abbildung 14 dargestellt zu haben. Auf den folgenden Seiten möchte ich noch zwei weitere Software-Technologien beschreiben, mit denen der Webentwickler

ebenfalls einen Webservice aufsetzen kann. Beginnen möchte ich mit JavaServer Pages (JSP) bzw. Java Servlets.

2.4 Java-Technologien zur Implementierung eines Webservice

Bei meiner eigenen Implementierung habe ich schon auf die JavaServer Pages bzw. Servlet-Technologie zurückgegriffen, als ich den Tomcat-Server verwendet habe, um den Sesame-Triplestore dort zu installieren. Dabei habe ich eine WAR-Datei (wird später erläutert) des Sesame-Java-Frameworks in einem Tomcat-Verzeichnis abgelegt, die zur Installation einer Software auf dem Tomcat-Server geführt hat, mit der der Entwickler verschiedene Triplestores verwalten kann und die gleichzeitig auch eine REST-Schnittstelle bereit stellt, mit der die konfigurierten Triplestores auf dem Tomcat-Server per SPARQL-Query in einem HTTP-GET-Request abgefragt werden können. Bei der Sichtung von Technologien für die Implementierung von Webservices ist mir allerdings aufgefallen, dass es für JSP keine vorhandene Klassenbibliothek gibt, mit der der Entwickler innerhalb einer JSP-Seite auf einen Sesame-Triplestore zugreifen kann. Stattdessen bin ich bei meiner Suche nur auf die Java Database Connectivity (JDBC) API gestoßen, mit der der Entwickler eine SQL-Datenbank in ein Servlet bzw. eine JSP-Seite einbinden kann. Würde ich eine relationale SQL-Datenbank für meine Implementierung verwenden, böte sich jedoch die Möglichkeit sowohl Symfony/Doctrine als auch JSP einzusetzen, um auf diese zugreifen zu können. JavaServer Pages, Symfony und auch das später noch zu erläuternde Ruby on Rails verfolgen, neben ihrer Kompatibilität mit relationalen Datenbanken auch einen ähnlichen Ansatz für die Implementierung dynamischer Inhalte in Webseiten. Die Version 1.0 von JSP wurde im Jahr 1999 veröffentlicht (Vgl. Wikipedia 2016d) und sollte ein verändertes Konzept, im Vergleich zu CGI oder Java Servlet, bieten, wie statische HTML- und CSS-Markups mit Programmiersprachen-spezifischen Schlüsselwörtern vermischt werden können. Mit JSP wurde das Problem, wie es bei CGI bestand, gelöst, dass während der Entwicklung der HTML-Quellcode direkt in den Quellcode der verwendeten Programmiersprache eingebettet werden musste (Bergsten 2003, Vgl. S. 4). Auch die Entwicklung von Java Servlets führt in der Regel dazu, dass verschiedene Bereiche einer Webanwendung auf ungünstige Weise miteinander vermischt werden. Außerdem ist es notwendig, den Servlet-Quellcode erneut zu kompilieren, wenn das Look & Feel der Anwendung verändert wird. Änderungen am Layout müssen manuell in Form von HTML- und CSS-Schlüsselwörtern in den Servlet-Quellcode eingefügt werden (Bergsten 2003, Vgl. S. 24). JSP hat daran schon 1999 etwas geändert. Mit JSP ist es möglich beides, Markups und Programmiersprachen-abhängige Schlüsselwörter in eine Datei zu schreiben und doch besser voneinander zu trennen. Eine JSP-Seite enthält nämlich, neben normalen HTML-Markups, spezielle JSP-Elemente, die es dem Server erlauben, dynamische Inhalte in die Webseite zu integrieren. Der Programmieransatz wird umgekehrt: Der Entwickler schreibt eine Webseite und fügt JSP-Elemente ein und nicht wie bei CGI oder Java Servlet, bei denen ein Programm geschrieben wird, das HTML- und CSS-Markups enthält und ausgibt. Wenn ein Client eine JSP-Seite abruft, führt der Server die JSP-Elemente in der Webseite aus, verbindet die Resultate mit dem statischen Teil der Webseite und sendet die dynamisch erstellte Webseite zurück an

den Client, bei dem sie im Browser angezeigt wird (Bergsten 2003, Vgl. S. 4). Dieses Konzept, dass die dynamischen Inhalten auf dem Server mit speziellen Schlüsselwörtern erzeugt werden und dann in die statischen HTML-Markups eingefügt werden, um dann das Ergebnis an den Aufrufer (in der Regel der Webbrowser) zurück zu schicken, findet sich auch bei PHP und folglich auch bei dem PHP-Framework Symfony. Abbildung 19 soll dieses Konzept verdeutlichen. Ein weiterer Vorteil von JSP ist, dass sie auf der Java Servlets-API

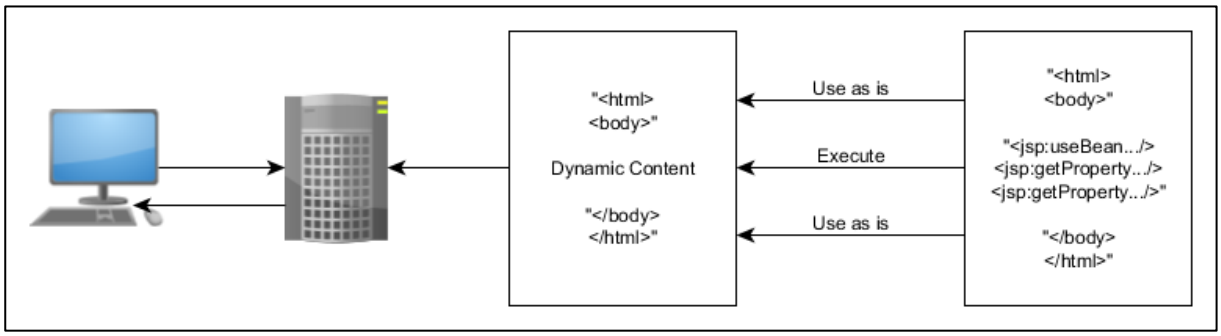

Abbildung 19: Generieren von dynamischen Inhalten mit JSP-Elementen (Bergsten 2003, Vgl. S. 4)

aufsetzen und damit Zugriff auf alle Enterprise Java APIs haben (Bergsten 2003, Vgl. S. 7). Der Entwickler kann also auf schon bestehende Schnittstellen und Funktionen zurückgreifen, wenn er eine JSP-Seite entwickeln möchte. Ferner sei noch angemerkt, dass JSP eine Spezifikation und kein Produkt ist, d. h, dass Hersteller verschiedene Implementierungen anbieten können, die im Idealfall zu besserer Performance und Qualität führen können (Bergsten 2003, Vgl. S. 9). Da JavaServer Pages auf der Java Servlet-Technologie beruhen, können sie auch deren Vorteile erben: Java Servlets sind plattform- und herstellerunabhängig. Sie sind in Java geschrieben und können so die Vorteile aller anderen Java-Technologien nutzen. Sie sind effizient, d. h. jede Anfrage an ein Servlet wird in einem separaten Thread innerhalb eines Prozesses ausgeführt, im Gegensatz zu CGI, wo für jede Anfrage ein neuer Prozess gestartet wird. Außerdem können Servlets Zugriff auf Ressourcen haben, die zwischen verschiedenen Anfragen innerhalb eines Prozesses gespeichert sind, wie bspw. Datenbankverbindungen. Servlets haben zusätzlich eine hohe Skalierbarkeit (Bergsten 2003, Vgl. S. 19). Servlets und damit JavaServer Pages laufen aber nicht direkt auf einem kompatiblen Webserver. Es ist eine Zwischenschicht notwendig, die Servlet Container genannt wird. Ein Servlet Container ist die Verbindung zwischen einem Webserver und den darauf laufenden Servlets. Er stellt die Laufzeitumgebung für alle Servlets auf dem Server bereit und ist verantwortlich für das Aufrufen und Laden der Servlets (Bergsten 2003, Vgl. S. 20). Außerdem ist er zuständig für das Mapping von eingehenden Anfragen zu einem registrierten Servlet. Anhand des aufgerufenen URI wird die Anfrage zu ihrem passenden Servlet weitergeleitet. Nachdem die Anfrage verarbeitet wurde, ist es die Aufgabe des Containers die Antwort in eine HTTP-Antwort-Nachricht umzuwandeln und diese zurück an den Client zu senden (Bergsten 2003, Vgl. S. 21). Innerhalb eines Containers wird jede Webanwendung durch einen Servlet Context repräsentiert. Der Servlet Context ist verbunden mit einem einheitlichen URI-Pfad, der Kontext-Pfad genannt wird. Diese URI-Pfade werden verwendet, um jede Anfrage zu einer entsprechenden Ressource weiterzuleiten (Bergsten 2003, Vgl. S. 22). Damit übernimmt der Servlet Container eine ähnliche Funktion wie die Controller-Klasse im Symfony-Framework. Außerdem kann ein

Servlet Context Objekte enthalten, die von allen Komponenten einer Anwendung geteilt werden, wie bspw. Datenbankverbindungen (Bergsten 2003, Vgl. S. 22). Servlet Container, die mindestens mit der Servlet 2.2 Spezifikation kompatibel sind, unterstützen die Web Application Archive (kurz WAR-Dateien). In diesen Archiven befinden sich alle Dateien, die eine Java Web-Applikation ausmachen. Eine Java Web-Applikation besteht typischerweise aus einer Kombination unterschiedlicher Arten von Ressourcen, wie JSP-Seiten, Servlets, statischen HTML-Seiten, Custom Tags, Bibliotheken und anderen Java-Klassen-Dateien. Zusätzlich zu diesen Dateien befindet sich in einer WAR-Datei auch ein Deployment Descriptor, der beschreibt, wie die unterschiedlichen Dateien der Webanwendung zusammen passen (Bergsten 2003, Vgl. S. 21). Wird eine HTTP-Anfrage an einen Webserver geschickt, auf dem eine JSP-Anwendung läuft, läuft die Verarbeitung nach dem folgenden Muster ab: Um alle Elemente in der Webseite verarbeiten zu können, wandelt der Container die JSP-Seite in ein Servlet um. Die dabei entstandene Java-Klassendatei wird auch JSP „page implementation class“ genannt. Alle Template-Text-Elemente (also alle Elemente, die kein JSP-Element sind) werden zu „println“-Statements umgewandelt und alle JSP-Elemente werden zu Java-Code konvertiert und daraufhin kompiliert, d. h. in Maschinensprache übersetzt. Dies wird als „Translation“-Phase bezeichnet. Der JSP-Container ist dann außerdem verantwortlich für das Aufrufen des JSP „page implementation class“, um Anfragen verarbeiten und Antworten darauf generieren zu können. Dies wird auch als „Request Processing“-Phase bezeichnet. Wenn die JSP-Seite verändert wird, wird zuerst wieder die „Translation“-Phase durchlaufen, bevor wieder die „Request Processing“-Phase gestartet wird (Bergsten 2003, Vgl. S. 26 - 27). Bleibt die JSP-Seite unverändert, werden alle Anfragen sofort vom Server ohne den Zwischenschritt des Kompilierens verarbeitet, was einen Geschwindigkeitszuwachs bedeuten kann (Bergsten 2003, Vgl. S. 6). Befindet sich die Webanwendung in dieser Phase arbeitet sie nach dem oben erwähnten MVC-Architekturmuster. Dabei findet eine Trennung von der Präsentation (View), der Anfragenentgegennahme und -verarbeitung (Controller) sowie der Daten und der Geschäftslogik (Model) statt. Die in dieser Masterarbeit vorgestellten Web-Anwendungstechnologien können alle auf das Model-View-Controller-Architekturmuster zurückgeführt werden. Dies gilt auch für die Webtechnologie, die ich im letzten Unterkapitel von Kapitel zwei vorstellen möchte: Ruby on Rails.

2.5 Ruby on Rails für die Implementierung eines Webservice

„Ruby on Rails funktioniert plattformübergreifend (…)“ (Tate and Hibbs 2007, S. IX) und ist ein Software-Framework. Ruby on Rails (RoR) verwendet die Model2-MVC-Variante, die die gleichen Prinzipien wie das schon erwähnte MVC-Architekturmuster hat, diese aber an zustandsfreie Webanwendungen anpasst (Tate and Hibbs 2007, Vgl. S. 1 u. 4). Eine Besonderheit von RoR ist die Möglichkeit Metaprogrammierungstechniken zu nutzen, bei denen Programme genutzt werden, „um andere Programme zu schreiben.“ (Tate and Hibbs 2007, Vgl. S. 2) Mit der Scaffolding genannten Technik „generiert [RoR] den Großteil des (…) benötigten [Software-] Gerüsts automatisch.“ So entfällt das Generieren von temporärem Code, mit dem der Entwickler sehen kann, wie die Komponenten und die von

ihm entwickelten Grundfunktionen zusammenarbeiten. „Rails erzeugt einfache automatische Tests (…) [und] stellt darüber hinaus unterstützenden Code zur Verfügung, der die Entwicklung und Ausführung von Tests vereinfachen kann.“ (Tate and Hibbs 2007, S. 3) „Scaffolding ist vor allem für Prototyping gedacht und wird in produktiven Anwendungen fast immer mit eigenem Code ergänzt.“ (Wikipedia 2016h) Nach Tate und Hibbs kann der Entwickler bei RoR außerdem „den gesamten Konfigurationscode gegenüber ähnlichen Java-Frameworks um den Faktor fünf oder mehr reduzieren, wenn er (…) den gängigen Konventionen [folgt] (…).“ (Tate and Hibbs 2007, Vgl. S. 3) RoR läuft sowohl auf dem vollständig in Ruby geschriebenen Webserver WEBrick als auch auf dem in dieser Arbeit schon erwähnten Webserver Apache. „Der Webserver leitet eingehende Requests an ein Ruby-Skript des Rails-Frameworks weiter, das Dispatcher genannt wird. Rails besitzt je einen Dispatcher (Verteiler) pro Webserver.“ Der Rails-Dispatcher verarbeitet die eingehende URL „und ruft die entsprechende Aktion auf dem entsprechenden Controller auf.“ (Tate and Hibbs 2007, S. 13) Damit übernimmt der Rails-Dispatcher eine ähnliche Funktion wie die Controller-Klasse im Symfony-Framework oder der Java Servlet Container: Er interpretiert die eingehenden Requests und veranlasst den Controller eine Antwort auf den Request zu generieren. „Die Controller-Aktion kann dann das Modell aufrufen und ruft abschließend einen View auf.“ (Tate and Hibbs 2007, S. 13) Abbildung 20 soll dies verdeutlichen:

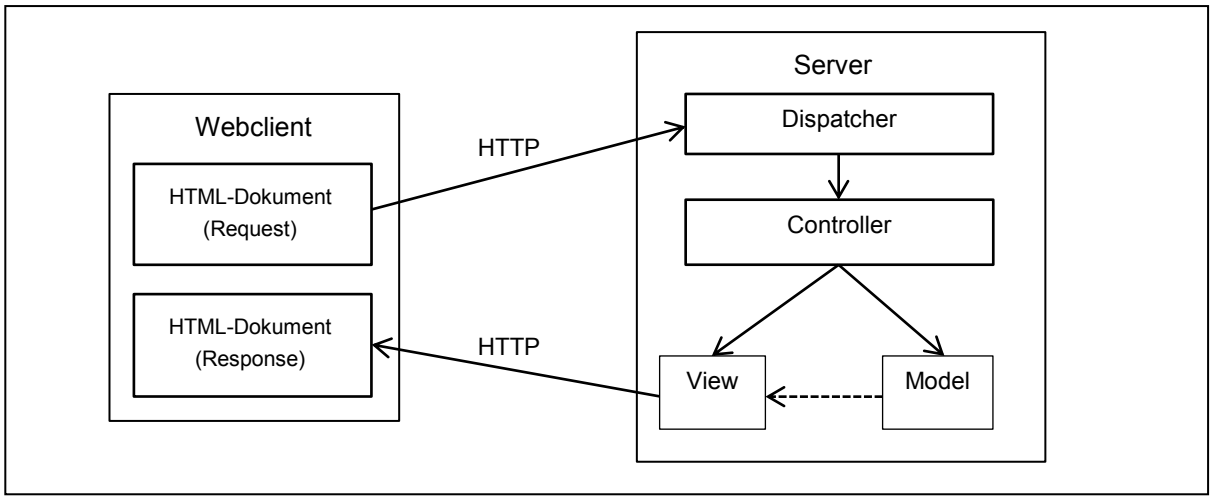

Abbildung 20: Der MVC-Fluss von Rails (Tate and Hibbs 2007, Vgl. S. 12)

Um einen View zu erstellen, wird in RoR ein Template verwendet. „Für Rails ist ein Template einfach eine HTML-Seite mit eingestreutem Ruby-Code.“ (Tate and Hibbs 2007, S. 14) Diese Art der Trennung von Darstellungs- und Formatierungsoptionen von der (dynamischen) Programmlogik findet sich sowohl bei Symfony mit der Sprache „Twig“ als auch bei JSP bei der Trennung von „Template“-Text und JSP-Elementen wieder. Damit in der View überhaupt sinnvolle Inhalte, die vom Request angefordert wurden, dargestellt werden können, müssen diese vom Modell (also den Daten) abgeholt werden. „Rails führt [dafür] das Active Record-Framework ein, das Objekte in der [verwendeten] Datenbank speichert [und wieder ausgibt].“ (Tate and Hibbs 2007, Vgl. S. 2) Ähnlich wie bei Symfony und der damit einhergehenden Nutzung des Doctrine Object Relational Mapper, bei dem eine Datenbanktabelle und ein dazugehöriges Objekt unabhängig voneinander aufgebaut und mit etwas PHP-Code, beim

sogenannten Mapping, aufeinander abgebildet werden, „verwendet [Active Record] hingegen keine Mapping-, sondern eine Wrapping-Strategie.“ Diese Unterscheidung zwischen Mapping und Wrapping habe ich nur bei Tate und Hibbs gefunden. Die offizielle RoR-Dokumentation im Netz (RoR-Documentation 2016) spricht bei Active Record auch nur von Object Relational Mapping. In der Web-Dokumentation von RoR wird ORM als eine Technik beschrieben, mit der Objekte von Klassen mit Tabellen eines relationalen DBMS verbunden werden, also die gleiche Vorgehensweise wie bei Symfony/Doctrine. Abbildung 21 soll

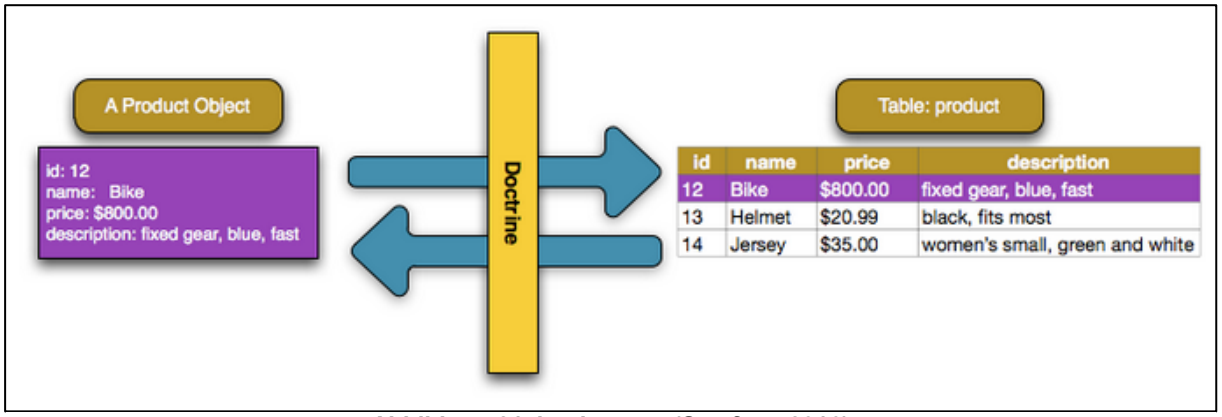

Abbildung 21, kopiert von (Symfony 2016)

diesen Ansatz nochmal ins Gedächtnis rufen. Beginnt ein Rails-Entwickler „mit dem Aufbau einer relationalen Datenbank, hüllt [er] jede Tabelle in eine Active Record-Klasse. Jede Instanz der Klasse steht [dabei] für eine Zeile der Datenbank.“ „Das Framework erkennt dann automatisch die Spalten der Datenbanktabelle und fügt sie dynamisch zur Active Record-Klasse hinzu.“ Der Vorteil der objektrelationalen Abbildung (ORM), die Unterstützung „viele[r] verschiedene[r] Arten von Datenbank-Schemata (…)“ (Tate and Hibbs 2007, S. 20), kann auch hier genutzt werden. Ein Unterschied zu relationalen Datenbanken ist aber der, dass „Active Record seine Identifier (Schlüssel) auf eine einzelne Datenbankspalte [beschränkt].“ (Tate and Hibbs 2007, S. 39) Bei meiner Recherche bin ich aber auf eine RoR-Erweiterung für Active Record gestoßen (Dr.Nic 2013), die es ermöglicht auch zusammengesetzte Primär- und Fremdschlüssel zu verwenden, damit ist es möglich das Entity-Relationship-Modell aus den Abbildungen 15 und 16 in die Active Record-Logik zu überführen und 1 zu 1, 1 zu M, M zu 1 sowie, in diesem Fall nicht von Bedeutung, N zu M-Beziehungen zu modellieren. Eine 1 zu M-Beziehung wird durch das Schlüsselwort „has_many“ beschrieben. „‚Has_many‘ ist die andere Seite einer ‚belongs_to‘-Beziehung (…)“ (Tate and Hibbs 2007, S. 42), mit der eine M zu 1-Beziehung ausgedrückt werden kann. Eine 1 zu 1-Beziehung wird durch die Schlüsselwörter „has_one“ und „belongs_to“ modelliert. „Die Entscheidung, ob ‚belongs_to‘ oder ‚has_one‘ verwendet wird, basiert darauf, wo der Fremdschlüssel liegt.“ (Tate and Hibbs 2007, S. 45) Die Klasse, die den Fremdschlüssel enthält, verwendet das Schlüsselwort „belongs_to“ und die Klasse mit dem Primärschlüssel verwendet „has_one“. N zu M-Beziehungen werden durch das Schlüsselwort „has_and_belongs_to_many“ ausgedrückt. Diese Beziehungen benötigen eine gesonderte Beziehungstabelle. In Active Record drückt „jede Zeile einer Beziehungstabelle (…) eine Beziehung mit Fremdschlüsseln aus, enthält sonst aber keine weiteren Daten.“ (Tate and Hibbs 2007, S. 46) Dieser kurze Einblick sollte zeigen, dass auch mit RoR eine

Webanwendung nach dem MVC-Architekturmuster möglich ist. Im Prinzip sollte ein Softwareprojekt bei der Auswahl einer geeigneten Technologie neben der Wartbarkeit und Pflege dieser, auch die damit verbundene Unterstützung durch die (Web-) Community berücksichtigen. Deshalb hängt es auch, wie in Unterkapitel 2.2 beschrieben, vom Ranking einer Programmiersprache ab, ob diese zum Einsatz kommen sollte. Ruby befindet sich in unterschiedlichen aktuellen Rankings nicht mehr in den Top Ten (Stand: März 2016) und könnte deshalb nicht die erste Wahl sein. Hat ein Projektteam sich für eine Programmiersprache entschieden und mit der Entwicklung begonnen, sollte, wie im Literaturbericht erwähnt, der Endanwender immer im Auge behalten werden, denn für ihn, als Kunden, ist dieses Produkt schließlich gemacht. Ob es Probleme bei der Nutzung der Software gibt, sollte schon früh durch den Einsatz von Prototypen und eine formative Usability-Evaluation in Erfahrung gebracht werden.
Damit komme ich zum nächsten großen Kapitel dieser Arbeit: der vergleichenden Usability-Evaluation meiner und der offiziellen Implementierung des re3data.org-Webservice.

3 Vergleichende Usability-Evaluation

Wie in den vorangehenden Kapiteln dargestellt, liegt das Hauptaugenmerk auf dem re3data.org-Webservice. An ihm habe ich mich bei meiner Implementierung orientiert und deshalb soll nun mein Webservice mit der offiziellen Webseite verglichen werden. Als Evaluationsmethoden bieten sich die Heuristische Evaluation und ein Usability-Test an. „Fu et al. (2002) geben als Handlungsempfehlung an, dass Heuristische Evaluationen und Usability-Tests kombiniert werden sollten, wobei die Heuristische Evaluation vor den Tests durchgeführt wird, damit die Nutzer sich auf die Wissensebene konzentrieren können (…) (vgl. Abs. 4.6)“ ((Fu, Salvendy, and Turley 2002) referenziert in (Sarodnick and Brau 2011, S. 208)) und damit getestet werden kann, ob das System einfach erlernbar ist. Wichtig zu wissen ist, dass ich keine vollständigen Evaluationen durchgeführt habe, so wie die Fachliteratur sie empfiehlt, denn das würde den Rahmen dieser Arbeit sprengen. Stattdessen habe ich nur exemplarisch eine Heuristische Evaluation, mit mir als Experten, durchgeführt und einen Usability-Test mit zwei Testpersonen. Auf den folgenden Seiten sollen deshalb immer nur das Prinzip und die Vorgehensweise beschrieben werden, wie diese beiden Evaluationsmethoden richtig angewendet werden. Gediga und Hamborg unterteilen die Evaluationsziele und -kriterien bzw. Fragestellungen, nach denen die zu untersuchenden Systeme beurteilt werden sollen in drei Gruppen: Die erste Gruppe der Evaluationskriterien fragt: „Which is better?“. Dabei werden zwei zu untersuchende Systeme nach bestimmten Kriterien miteinander verglichen. Die zweite Gruppe beschäftigt mit der Frage: „How good?“ „Bei dieser Fragestellung geht es um die Ausprägung bestimmter Systemeigenschaften.“ In der dritten Gruppe von Evaluationskriterien wird gefragt: „Why bad?“ Diese wird vor allem für die formative Evaluation eingesetzt und soll Schwachstellen des Systems aufdecken, „die als Ausgangspunkt für die weitere Gestaltung dienen.“ ((Gediga and Hamborg 2002) referenziert in (Sarodnick and Brau 2011, S. 121)) Da ich in diesem Kapitel die Usability zweier Systeme messen und damit auch vergleichen möchte, gehören meine Evaluationskriterien sowohl bei der Heuristischen Evaluation als auch beim

später zu beschreibenden Usability-Test zur ersten Gruppe. Bevor ich mit der Beschreibung der Heuristischen Evaluation der beiden Webservices beginne, möchte ich anhand von Screenshots deren unterschiedliche Oberflächen erklären. Dabei zeige ich zuerst die Oberfläche der offiziellen Webseite, um danach meine Implementierung zu beschreiben:

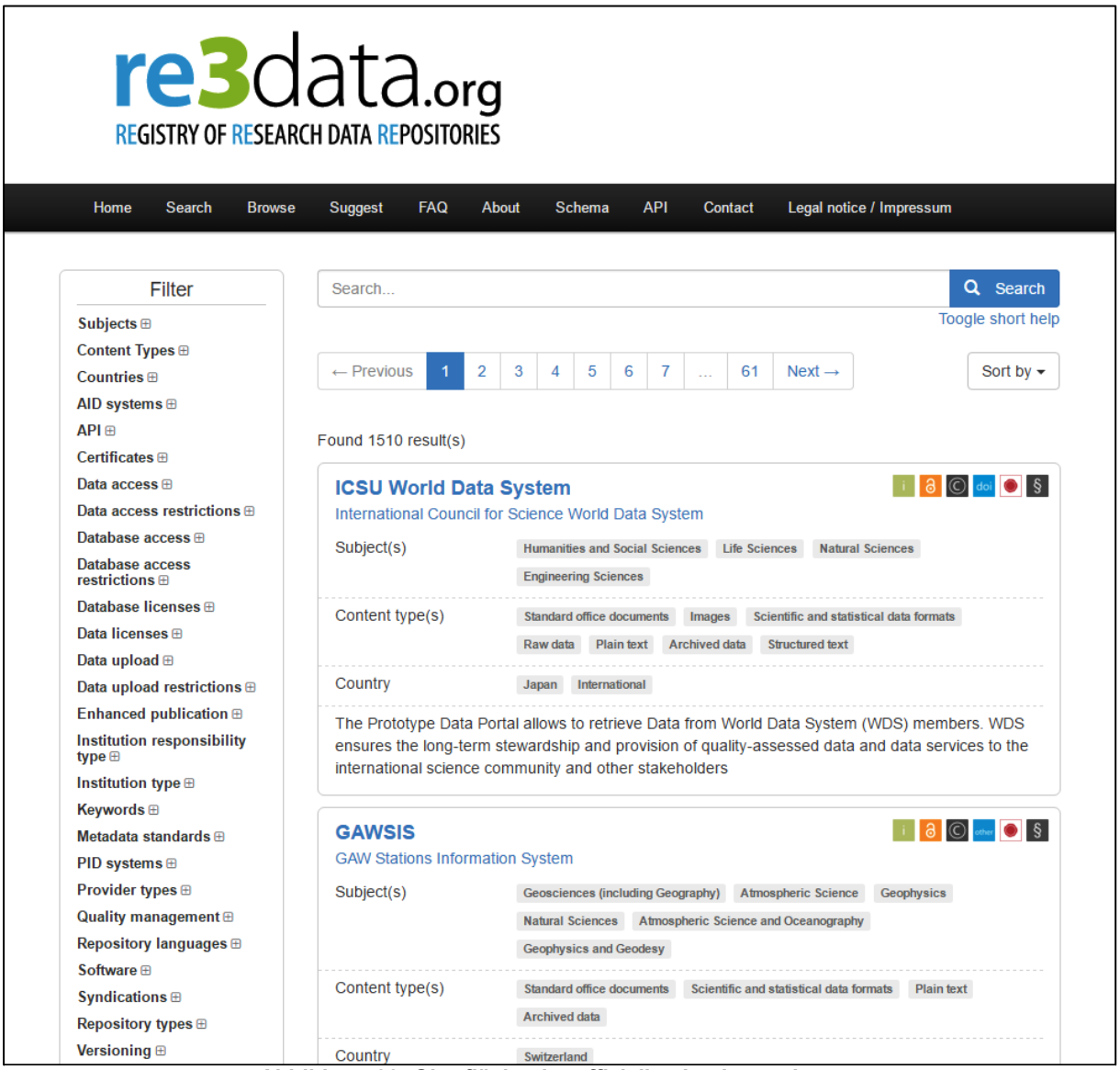

Abbildung 22: Oberfläche der offiziellen Implementierung

Wie Abbildung 22 zeigt, gibt es auf der linken Seite eine Seitenleiste, die die Facetten der einzelnen Repositorien widerspiegelt. Klickt der Nutzer auf Fadenkreuz rechts neben dem Facettenname, öffnen sich die Suchkriterien dieser Facette. Auf diese kann der Nutzer dann klicken, um seine Suche einzugrenzen. Das ausgewählte Suchkriterium kann mit anderen kombiniert werden und wird fett gedruckt angezeigt. Eine alternative Suchstrategie bietet der Suchschlitz über dem Paginationsbalken. Dort kann der Nutzer beliebige Suchkriterien eingeben. Dabei wird aber nicht deutlich, welches Suchkriterium im Suchschlitz für einen Treffer verantwortlich ist. Unter der Gesamtzahl der Treffer befindet sich die Übersichtsseite der gefundenen Repositorien. Jeweils rechts neben dem Repositoriennamen befinden sich Icons, die symbolisieren, welche Suchkriterien das Repositorium unterstützt: bspw. das orangene Schlosssymbol für ein Open Access-Repositorium. Je mehr Icon-Kriterien ein

Repositorium erfüllt, desto höher wird es gerankt und hat so eine höhere Gewichtung bei der Rangfolge der Anzeige.

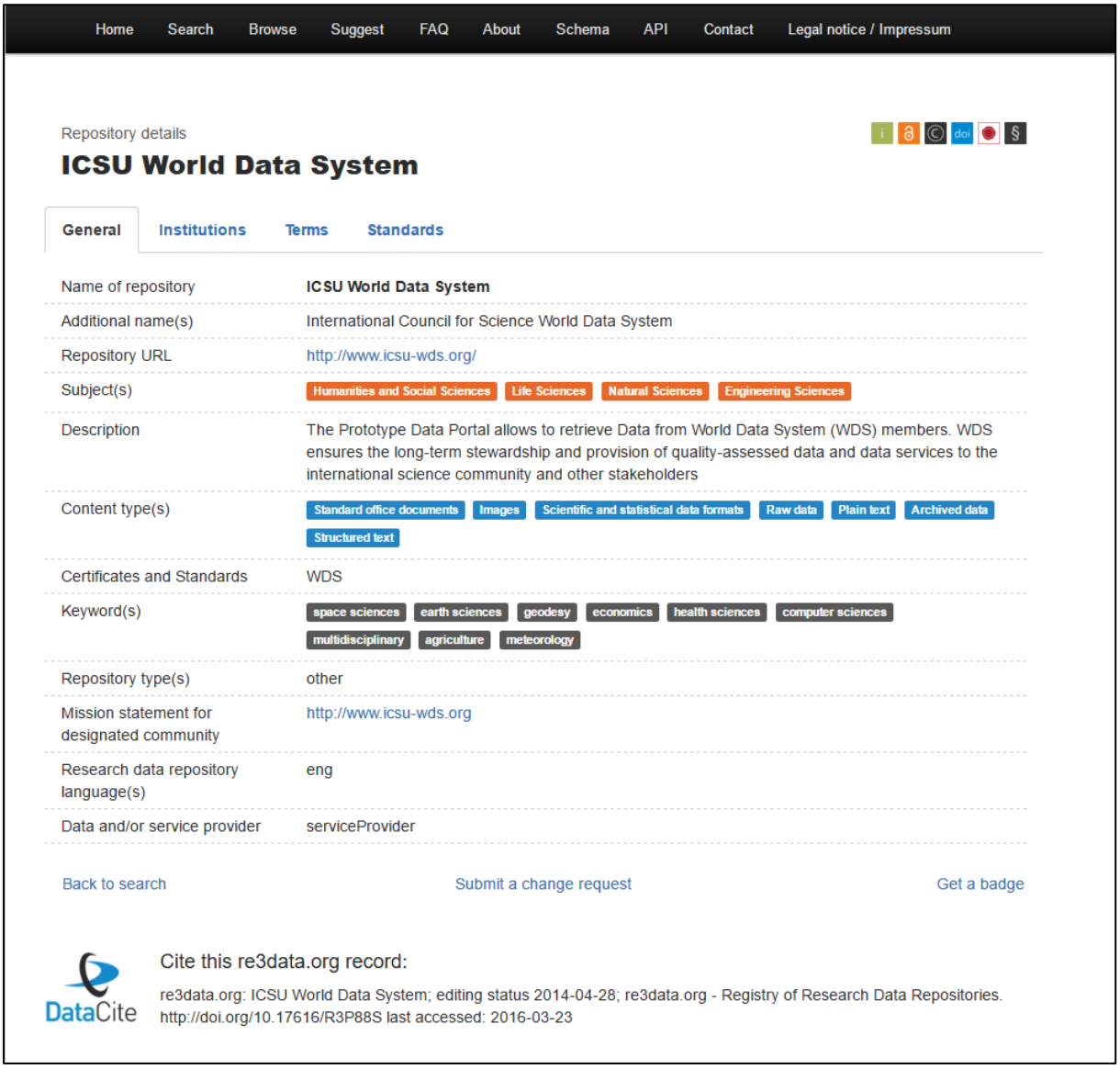

Abbildung 23: Detailansicht bei der offiziellen Implementierung

Klickt der Nutzer auf einen Repositoriennamen, öffnet sich ein neuer Browser-Tab, in dem sich die Detailansicht befindet. Dort hat der Nutzer über Tabs innerhalb der Seite die Möglichkeit, verschiedene Aspekte wie „Institutions“, „Terms“ oder „Standards“ auszuwählen. Diese Struktur habe ich auch für meine Implementierung übernommen, die ansonsten aber eher dem Muster der alten offiziellen Implementierung, mit einigen Unterschieden, folgt (siehe Abbildung 24). Es gibt dort zwar auch einen Sidebar, der enthält aber nur Checkboxen, die unterschiedliche Facetten eines Repositoriums auswählbar machen, bspw. ob der Datenupload für Forschungsdaten „open“ oder „restricted“ sein soll. Außerdem gibt es noch sechs Dropdown-Menüs, mit denen unterschiedliche Suchkriterien eingestellt werden können. Hat der Benutzer über ein Dropdown-Menü ein Suchkriterium ausgewählt, erscheint dieses als Label unter dem Dropdown-Menü und kann dort auch wieder gelöscht werden. Der Nutzer kann ferner auf der Übersichtsseite auf die Labels eines Repositoriums klicken, um diese als Suchkriterium auszuwählen. Diese Möglichkeit besteht auch bei der offiziellen

Implementierung, wobei dort nicht über ein farbiges Label die Auswahl sichtbar gemacht wird. Die Auswahl erscheint stattdessen als Kriterium in der facettierten Suche. Abbildung 24 zeigt die Übersichtsseite meiner Implementierung.

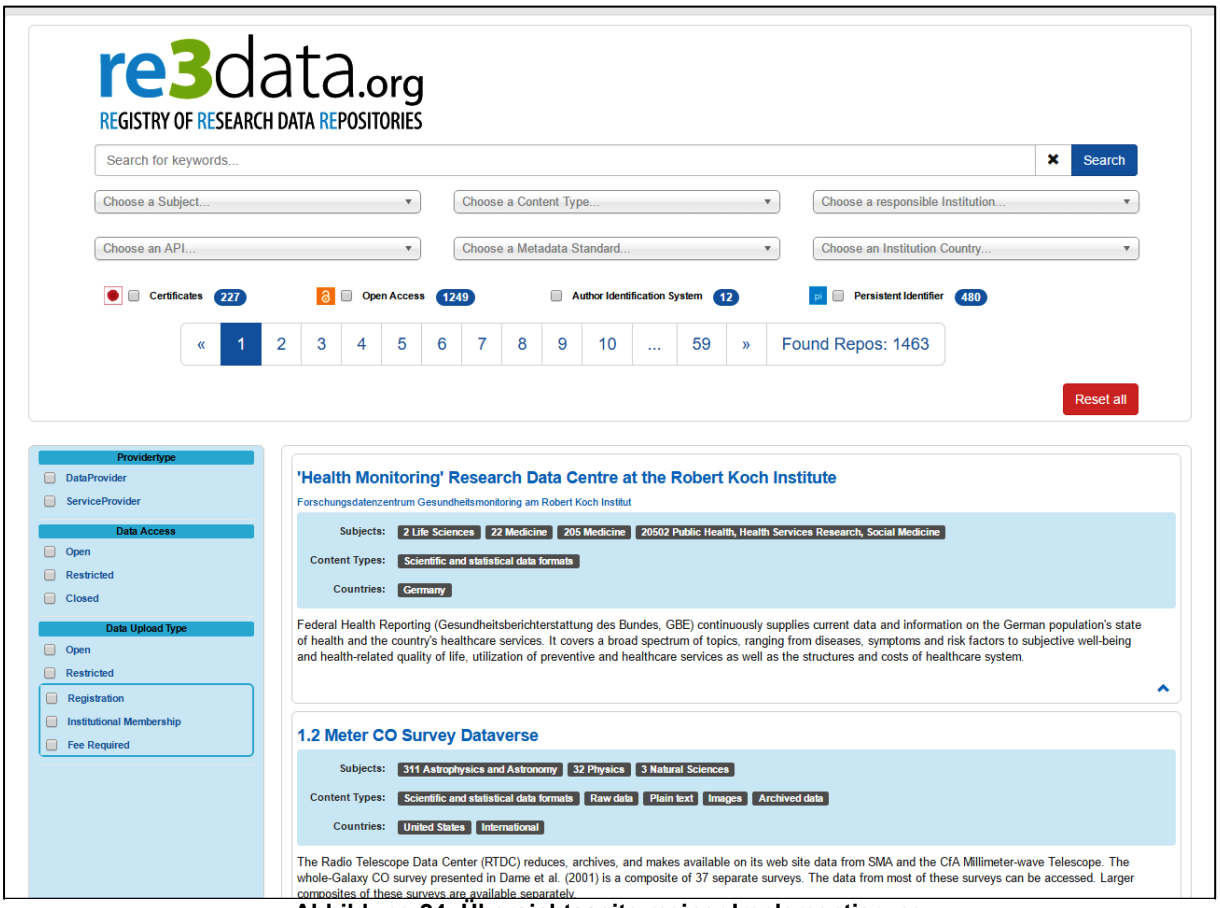

Abbildung 24: Übersichtsseite meiner Implementierung

Dort kann der Nutzer auch über die Checkboxen unter den Dropdown-Menüs weitere Suchkriterien auswählen. Das Suchkriterium „Open Access“ gibt es nicht direkt bei der offiziellen Implementierung, da Open Access-Repositorien immer höher gerankt werden als Repositorien ohne dieses Merkmal. Die Detailansicht kann bei meiner Implementierung aufgerufen werden, wenn der Nutzer auf einen Repositoriumsnamen klickt. Dabei wird die Detailansicht nicht in einem neuen Browser-Tab geöffnet, sondern diese erscheint an der Stelle, wo die Übersicht über die Repositorien angezeigt wird. Der Nutzer kann zwischen der Detailansicht und der Übersichtsseite über die Browser-Buttons für das Vorwärts- und Zurückspringen auf eine Seite wechseln. Das Design der Detailansicht unterscheidet sich dabei, wie weiter oben erwähnt, nicht von dem der offiziellen Webseite und wird deshalb hier nicht separat dargestellt. Da ich nun die Unterschiede im Layout der beiden Seiten dargestellt habe, komme ich zum ersten Teil meiner vergleichenden Evaluation: der Heuristischen Evaluation.

3.1 Die Heuristische Evaluation: Vergleich zweier Webservices

Die Heuristische Evaluation wird immer von Usability-Experten durchgeführt. Für die Heuristische Evaluation in dieser Arbeit bin ich der Experte. Nielsen wies 1992 daraufhin, dass die Evaluatoren sowohl Domänenexpertise, Wissen über die Zielgruppe und die

Fähigkeit sich in die Perspektive des Nutzers hineinversetzen zu können mitbringen sollten, als auch domänenunabhängige Usability Expertise sowie Vorerfahrungen aus vergangenen Evaluationen. Mit dieser Kombination können die Experten 60 % Fehleraufdeckung erreichen (Sarodnick and Brau 2011, Vgl. S. 144 - 145). Findet ein Experte einen Verstoß gegen die Heuristiken, kann diese Erkenntnis auf ein Usability-Problem hindeuten. „Es besteht aber keine Kausalität zwischen Verstoß und Usability-Problem, da stets der Kontext der Nutzung zu betrachten ist." (Sarodnick and Brau 2011, S. 144) Um Heuristiken für eine Evaluation besser an eine Anwendungsdomäne anpassen zu können, bietet sich einerseits die Möglichkeit neue Heuristiken in eine bestehende Heuristik aufzunehmen oder die bestehenden Heuristiken zu erweitern. Diese domänenspezifische Anpassung kann dazu führen, dass mehr Usability-Probleme aufgedeckt werden (Sarodnick and Brau 2011, Vgl. S. 148). Dies mache ich aber bei meiner Evaluation nicht, denn, wie schon im Literaturbericht erwähnt, greife ich dabei nur auf die Heuristiken des BibEval-Kriterienkataloges zurück. Aus diesem Kriterienkatalog habe ich 36 Fragen bzw. Kriterien ausgewählt, die an das zu untersuchende System gestellt werden. Die Fragen kommen aus den Bereichen „Information & Kommunikation" und „Recherche im Bestand" sowie aus den Unterkategorien „Seitenüberblick", „Suchen & Erkunden" und „Präsentation & Zugriff". Im Folgenden möchte ich die Begründungen für die Auswahl der 36 Fragen aufzeigen. Diese befinden sich in der folgenden Tabelle 2:

Ausgewählte Kriterien	***Begründung für die Auswahl***
Information & Kommunikation - Allgemein	
3. Sind die Inhalte in allen für das jeweilige Zielpublikum relevanten Sprachen vorhanden?	Damit ein Webservice sein Zielpublikum erreichen kann, sollte er auch die Sprache des Zielpublikums anbieten. Englisch als Hauptsprache in der internationalen Wissenschaftskommunikation ist deshalb Pflicht.
Information & Kommunikation - Seitenüberblick	
1. Sind die Navigationselemente als solche klar erkennbar?	Sind die Navigationselemente gut sichtbar, wirkt sich das positiv auf die Usability und damit die Bedienung der Webseite aus.
2. Ermöglichen die verwendeten Navigationselemente einen schnellen Überblick über die der Seite zugrundeliegende Struktur und die zur Verfügung stehenden Funktionalitäten?	Dies sollte der Fall sein, damit die Bedienung der Webseite für das Zielpublikum kein Hindernis darstellt.
Recherche im Bestand - Allgemein	
2. Stehen dem Anwender jederzeit angemessene Extraktionsmechanismen (z.B. Email, Print, Speichern, etc.) zur weiteren Verwertung seiner Recherchen und Treffer zur Verfügung?	Diese Möglichkeit unterstützt eine Vielzahl von unterschiedlichen Arbeitsstrategien, denn der Nutzer muss nicht immer „online" sein, um auf die Informationen des Webservice zugreifen zu können. Außerdem kann der Nutzer bei angemessenen Extraktionsmöglichkeiten, seine Suchergebnisse besser mit anderen teilen, z. B. via E-Mail.
Recherche im Bestand - Suchen & Erkunden	
1. Ermöglichen die angebotenen Rechercheoptionen die Unterstützung unterschiedlicher Suchstrategien?	Gibt es mehrere Möglichkeiten der Bedienung werden dadurch unterschiedliche Nutzergruppen, die unterschiedlich eine Webseite benutzen, abgedeckt.
2. Ist während einer Recherche jederzeit ein Wechsel zwischen den	Dieses Kriterium erlaubt mehr Flexibilität bei der Suche und mehr Möglichkeiten seine

unterschiedlichen Suchstrategien möglich?	Suchstrategie im Nachhinein noch anzupassen.
Recherche im Bestand - Suchen & Erkunden - Einfache Suche	
1. Kann die Suche sowohl durch das Drücken eines dedizierten Buttons als auch per "Enter" gestartet werden?	Unterschiedliche Ansätze und Gewohnheiten bei der Bedienung einer Webseite sollten abgedeckt werden.
2. Beherrscht die Suche eine unterschiedliche Anzahl an Suchbegriffen?	Eine unterschiedliche Anzahl von Suchbegriffen führt dazu, dass es unterschiedliche Grade der Verfeinerung einer Suchanfrage geben kann.
3. Ist das Eingabefeld breit genug, um mehrere Suchbegriffe gleichzeitig anzeigen zu können?	Der Nutzer sollte immer sehen, für welche Suchkriterien und -begriffe er sich entschieden hat. Klarheit in dieser Frage bietet ein besseres Feedback für den Nutzer und ein besseres Verständnis der Suchergebnisse.
4. Ist es für den Nutzer ersichtlich, wie mehrere Suchbegriffe miteinander verknüpft werden (welche Operator wird per default verwendet)?	Klarheit in dieser Frage bietet ein besseres Feedback für den Nutzer und ein besseres Verständnis der Suchergebnisse.
Recherche im Bestand - Suchen & Erkunden - Erweiterte Suche	
3. Ist die Sortierung innerhalb von Drop-Down-Menüs intuitiv verständlich (alphabetisch, nach Relevanz)?	Ein besseres Verständnis der Oberfläche und seiner Bedienelemente führt zu einer besseren Nutzung dieser.
Recherche im Bestand - Suchen & Erkunden - Eingabemöglichkeiten	
1. Werden die Booleschen Operatoren AND, OR und NOT unterstützt?	Gerade in der Informationswissenschaft sollte das Verständnis dieser Operatoren vorhanden sein. Ein Informationssystem sollte also eine Eingabemöglichkeit für diese Operatoren bieten.
2. Ist es möglich, mehrere Operatoren gleichzeitig zu verwenden und ist klar, in welcher Reihenfolge diese verarbeitet werden, sofern der Anwender keine Klammern setzt?	Die Verwendung von und das Verständnis für diese Operatoren sollte das Informationssystem ermöglichen und erleichtern, indem es transparent macht, wie die Operatoren eingesetzt werden können und welche Auswirkungen dies hat.
3. Werden die Operatoren in unterschiedlichen Schreibweisen unterstützt?	Mehr alternative Möglichkeiten von Schreibweisen der Suchoperatoren vergrößern die Abdeckung der und das Verständnis bei der Nutzergruppe.
4. Können Wildcards, Trunkierungen und Klammern eingesetzt werden?	Nutzer mit Vorwissen im Bereich der regulären Ausrücke können das Informationssystem besser nutzen, indem sie ihr Expertenwissen verwenden können.
5. Ist es möglich Sonderzeichen und Umlaute in einer Anfrage zu formulieren und werden diese korrekt interpretiert?	Befinden sich in den Metadaten solche Zeichen ist es wichtig, dass das System diese auch bei der Suche unterstützt, damit die Metadaten auch gefunden werden können.
Recherche im Bestand - Suchen & Erkunden - Assistierende Funktionen bei der Suche	
1. Wird eine Suchhistorie angeboten, mittels der der Anwender frühere Suchanfragen wiederholt durchführen kann?	Dieses Feature bietet eine Erinnerungsfunktion für den Nutzer und unterstützt ihn damit bei seiner Suche.
3. Ist es möglich, frühere Anfragen aus der Suchhistorie zu editieren und dann neu abzuschicken?	Die Wiederverwendbarkeit und Nachnutzung von Suchanfragen erhöht den Komfort und die Geschwindigkeit bei der Suche, da der Nutzer die Suchanfragen nicht neu eingeben muss und sie auch nicht erinnern muss.
5. Wird eine Rechtschreibkorrektur angeboten und funktioniert diese korrekt?	Eine Rechtschreibkorrektur hilft dem Nutzer dabei korrekte Suchkriterien einzugeben. Dies kann zu besseren Suchergebnissen führen.
6. Werden die Vorschläge zur Rechtschreibkorrektur in der Nähe des Sucheingabefelds platziert, das den Tipp- oder Rechtschreibfehler beinhaltet?	Die geforderte Nähe zum Eingabefehler führt zu einer besseren Erkennbarkeit und Korrektur dessen.

9. Wird eine Autocomplete-Funktion in den Eingabefeldern zur Verfügung gestellt und scheinen die Vorschläge sinnvoll?	Autovervollständigung der Suchanfragen kann zu besseren Suchergebnissen führen, da der Nutzer erfahren kann, was das Informationssystem noch enthält.
Präsentation & Zugriff - Allgemein	
1. Unterstützt die Dialoggestaltung (z.B. die verwendeten Navigationselemente) einen einfachen Wechsel zwischen Suche, Trefferlisten und Detailanzeigen?	Dieses Kriterium ist wichtig, da nicht nur das einfache Finden von Treffern möglich sein sollte, sondern auch deren Auswahl und Anzeige ihrer vollständigen Daten.
Präsentation & Zugriff - Darstellung der Trefferliste	
1. Ist die Suchanfrage, die zur Ergebnisliste geführt hat, weiterhin vollständig sichtbar und kann sie direkt editiert werden?	Dies ist notwendig, damit der Nutzer seine Ergebnisse weiter verfeinern und eingrenzen kann.
2. Ist die Anzahl der Treffer gut sichtbar und in der Nähe der Ergebnisliste platziert?	Der Nutzer weiß dadurch, wie groß die Treffermenge ist und kann sie dann ggf. noch weiter einschränken. Dies wird ihm transparent durch dieses Kriterium ermöglicht.
3. Werden alle relevanten Informationen angezeigt oder besteht ein direkter Zugriff (z.B. per Mouseover-Effekt) auf diese Informationen?	Dies erleichtert es dem Nutzer zu bewerten, ob ein Treffer für ihn relevant ist. Sind weiterführende Informationen verborgen, kann dies die Bewertung der Suchergebnisse behindern.
4. Ist klar ersichtlich, ob es detailliertere Informationen zu den einzelnen Treffern gibt und wie auf diese Detailinformationen zugegriffen werden kann?	Ohne diese Informationen kann der Nutzer nicht seine Trefferliste endgültig bewerten. Der Nutzer muss wissen, wie er die Detailinformationen sichtbar machen kann.
5. Ist die anfängliche Sortierung der Treffer (Ranking) für den Anwender transparent?	Dieses Kriterium hilft dem Nutzer bei der Bewertung seiner Trefferliste und kann ihm helfen zu erkennen, wie diese zustande gekommen ist.
6. Ist es transparent, in wie weit und weshalb die einzelnen Treffer der Suchanfrage entsprechen?	Mit diesen Informationen kann der Nutzer erkennen, welche Suchkriterien er verändern muss, um seine Treffer zu verbessern. Er bekommt sozusagen Feedback zu seinen Suchkriterien.
Präsentation & Zugriff - Darstellung der Detailansicht	
1. Werden alle relevanten Informationen der Trefferliste ebenso angezeigt, wie zusätzliche Informationen, die hilfreich sind?	Dieses Kriterium hilft dem Nutzer bei der Bewertung seiner Trefferliste und damit seiner Suchstrategie.
4. Ist es möglich, direkt zwischen den Detailansichten der einzelnen Treffer zu navigieren?	Dieses Kriterium ermöglicht dem Nutzer einen besseren Überblick über seine Treffer und vereinfacht deren Bewertung.
6. Werden Extraktionsmechanismen wie Email, Bookmark und Drucken angeboten?	Extraktionsmechanismen fördern die Weiternutzung und Distribution der Suchergebnisse.
Präsentation & Zugriff - Manipulation der Trefferliste	
1. Ist es möglich, die Anzahl der Treffer pro Seite anzupassen, sofern die Trefferliste über mehrere Seiten paginiert ist?	Dieses Kriterium kann zu einer besseren Nutzung und Übersichtlichkeit der Trefferliste führen, da der Nutzer selbst entscheiden kann, wieviel Treffer er auf einmal bewerten möchte.
2. Ist es möglich, die Trefferliste nach unterschiedlichen Kriterien zu sortieren?	Unterschiedliche Kriterien zum Sortieren von Trefferlisten erhöhen die Abdeckung unterschiedlicher Nutzergruppen, da der Nutzer die Präsentation der Ergebnisse beeinflussen kann und so schneller zum Ziel gelangen kann.
3. Stehen dem Anwender Filterfunktionen zur weiteren Manipulation der Trefferliste (z.B. 'faceted search') zur Verfügung?	Dieses Kriterium führt zu einer besseren Filterung der Suchergebnisse und erleichtert damit dem Nutzer die Suche nach geeigneten Treffern.
4. Ist es möglich, Sortierungen und Filter wieder zu entfernen, um zum ursprünglichen Zustand der Trefferliste	Das Entfernen und das Zurücksetzen von Suchkriterien können dabei helfen, eine neue Suche zu beginnen oder den Suchradius zu

zurückzukehren?	erhöhen.
5. Ist es möglich, direkt zu einem bestimmten Treffer zu springen, bzw. im Fall von paginierten Trefferlisten direkt auf eine bestimmte Seite der Trefferliste zu navigieren?	Dieses Feature ist besonders wichtig, wenn die Anzahl der Treffer sehr hoch ist. Ein Paginationsbalken kann dabei helfen, sich effizient durch große Trefferlisten zu bewegen.

Tabelle 2: 36 Kriterien und die Begründung für ihre Auswahl

Hinzuzufügen ist, dass der von mir erstellte BibEval-Kriterienkatalog insgesamt 69 Kriterien enthielt. 33 Kriterien habe ich aber wieder verworfen und für diese die Bewertung als „nicht zutreffend“ ausgewählt. Beginnen möchte ich mit einem Vergleich der gefundenen Usability-

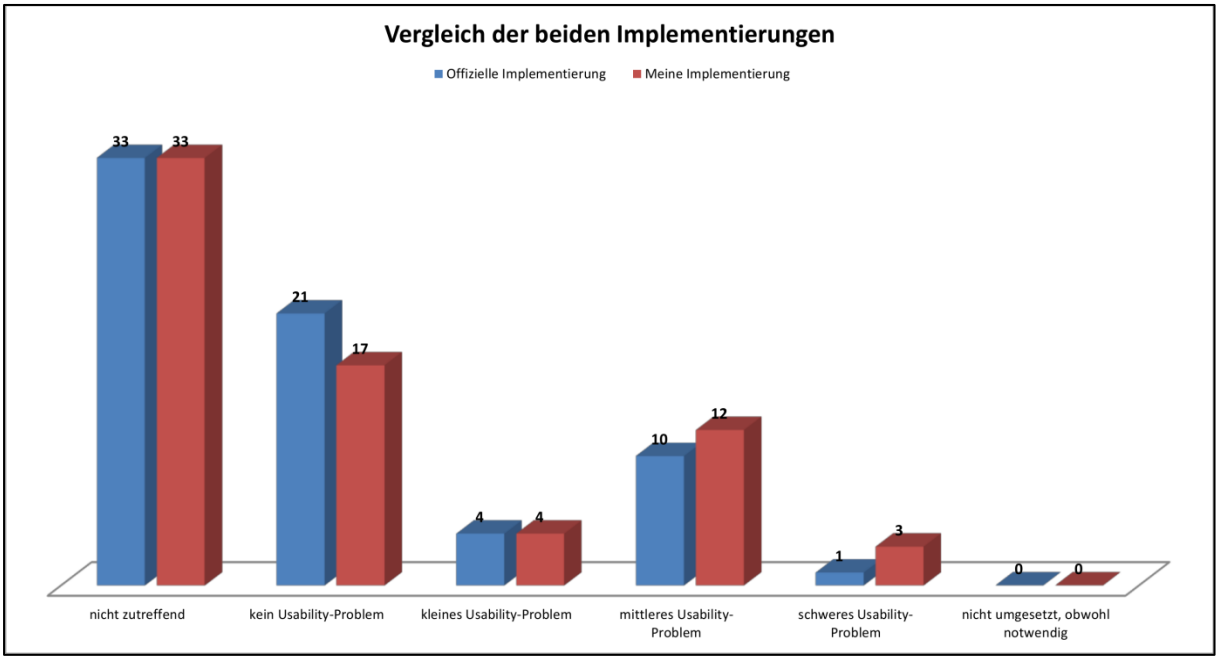

Abbildung 25: Vergleich der Implementierungen

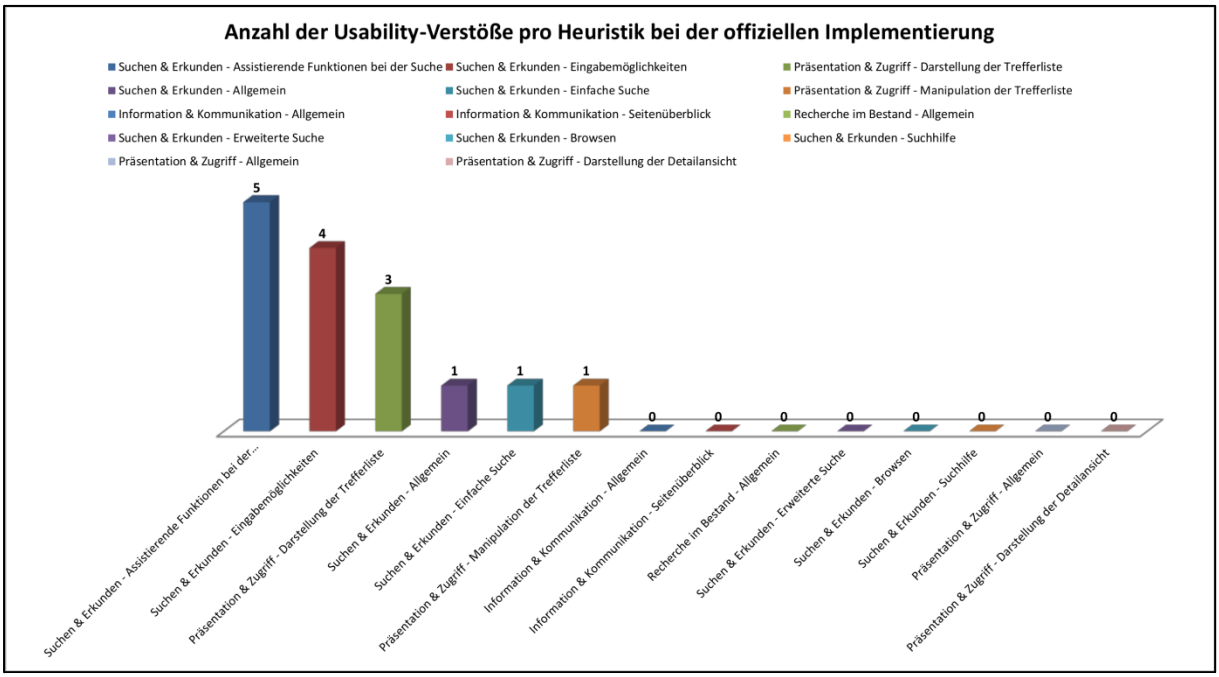

Abbildung 26: Usability-Verstöße pro Heuristik bei der offiziellen Implementierung

Probleme auf der offiziellen Webseite und bei meiner Implementierung (siehe Abbildung 25).

Abbildung 26 zeigt, dass ich bei der offiziellen Implementierung insgesamt 15 Usability-Probleme gefunden habe, die sich auf insgesamt 6 Heuristik-Kategorien verteilen, wobei 9

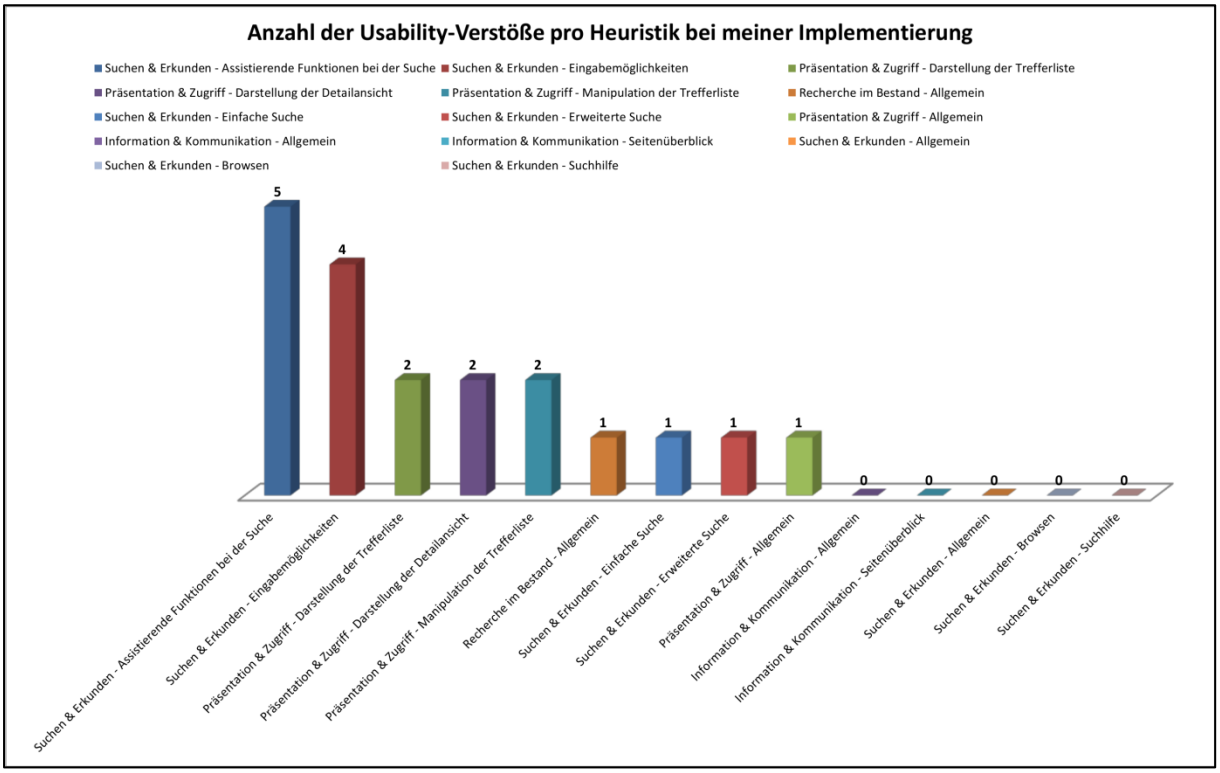

Abbildung 27: Usability-Verstöße pro Heuristik bei meiner Implementierung

Probleme auf die Oberkategorie „Suchen & Erkunden“ zutreffen. Zum Vergleich folgt die Verteilung der gefundenen Usability-Probleme bei meiner Implementierung in Abbildung 27. Hier kann man erkennen, dass wie bei der offiziellen Implementierung besonders viele Usability-Probleme bei der Heuristik-Kategorie „Suchen & Erkunden“ von mir gefunden wurden. Insgesamt habe ich weniger Usability-Probleme bei der offiziellen Implementierung gefunden, gerade auch bei den mittleren und schweren Usability-Problemen, wie es Abbildung 25 zeigt. Abbildung 27 zeigt außerdem, dass die Usability-Probleme bei meiner Implementierung auf mehr Heuristiken verteilt sind, als bei der offiziellen Implementierung. Beim Vergleich der gefundenen Usability-Probleme habe ich herausgefunden, dass teilweise die beiden Implementierungen die gleichen Probleme aufweisen. In der folgenden Tabelle 3 sind diese Probleme und ihre zugehörigen Heuristiken aufgelistet.

Ausgewählte Kriterien	***Ergebnis der Bewertung***
Recherche im Bestand - Suchen & Erkunden - Einfache Suche	
4. Ist es für den Nutzer ersichtlich, wie mehrere Suchbegriffe miteinander verknüpft werden (welche Operator wird per default verwendet)?	Die Verknüpfung ist für den Nutzer nicht offensichtlich. Es wird außerdem nicht klar, welcher Operator per Default verwendet wird. Ich bewerte dies als ein schweres Usability-Problem bei beiden Implementierungen, da nicht ersichtlich ist, wie die Suchergebnisse in diesem Fall zustande kommen.
Recherche im Bestand - Suchen & Erkunden - Eingabemöglichkeiten	
1. Werden die Booleschen Operatoren AND, OR und NOT unterstützt?	Die Operatoren werden nicht unterstützt. Da die Unterstützung nur für einen Experten in der Regel relevant ist, bewerte ich dieses Problem als ein mittleres Usability-Problem.
2. Ist es möglich, mehrere Operatoren gleichzeitig zu verwenden und ist klar, in	Dieses Kriterium baut auf dem vorangehenden auf. Da die Operatoren nicht unterstützt kann auch

welcher Reihenfolge diese verarbeitet werden, sofern der Anwender keine Klammern setzt?	dieses Kriterium nicht erfüllt werden. Ich bewerte es als ein mittleres Usability-Problem.
3. Werden die Operatoren in unterschiedlichen Schreibweisen unterstützt?	siehe 2.)
4. Können Wildcards, Trunkierungen und Klammern eingesetzt werden?	Diese Möglichkeiten sind nicht gegeben. Ich bewerte dies als ein mittleres Usability-Problem, da dies Nutzer mit Expertenkenntnissen über bestimmte Suchstrategien ausschließt.
Recherche im Bestand - Suchen & Erkunden - Assistierende Funktionen bei der Suche	
1. Wird eine Suchhistorie angeboten, mittels der der Anwender frühere Suchanfragen wiederholt durchführen kann?	Eine Suchhistorie als Bestandteil des Webservice ist bei beiden Implementierungen nicht direkt vorhanden. Nur im Suchschlitz der Freitextsuche kann, wenn der Webbrowser dies unterstützt, auf die Einträge zurückgegriffen werden. Allerdings gilt dieses Feature nicht für die anderen Einstellungsmöglichkeiten der Suche. Da dieses Kriterium bei beiden Implementierungen nicht vorhanden ist und nur ungenügend über die Browserfunktionen abgedeckt wird, bewerte ich dies als ein mittleres Usability-Problem.
3. Ist es möglich, frühere Anfragen aus der Suchhistorie zu editieren und dann neu abzuschicken?	Dies ist nur möglich über die Suchhistorie des Browsers, wenn dieser Formulareinträge speichert, und auch nur im Suchschlitz für die Freitext-Suche. Ansonsten unterstützen die beiden Seiten dieses Feature nicht. Da nach meiner Einschätzung dieses Feature aber nicht besonders wichtig ist, habe ich es nur als ein kleines Usability-Problem bewertet, da es viel leichter ist, einfach eine neue Suche zu starten, anstatt eine alte zu editieren.
5. Wird eine Rechtschreibkorrektur angeboten und funktioniert diese korrekt?	Da keine Rechtschreibkorrektur angeboten wird und ich dieses Feature als nicht vorrangig betrachte, bewerte ich das Fehlen als ein mittleres Usability-Problem.
6. Werden die Vorschläge zur Rechtschreibkorrektur in der Nähe des Sucheingabefelds platziert, das den Tipp- oder Rechtschreibfehler beinhaltet?	siehe 5.)
9. Wird eine Autocomplete-Funktion in den Eingabefeldern zur Verfügung gestellt und scheinen die Vorschläge sinnvoll?	Es wird keine Autocomplete-Funktion angeboten, was ich als ein mittleres Usability-Problem bewerte.
Präsentation & Zugriff - Darstellung der Trefferliste	
1. Ist die Suchanfrage, die zur Ergebnisliste geführt hat, weiterhin vollständig sichtbar und kann sie direkt editiert werden?	Wieder lässt sich die Suchanfrage nur im Suchschlitz editieren, falls die Formulardaten gespeichert wurden. Ansonsten sind alle Einstellungsmöglichkeiten über die Webseite weiterhin veränderbar. Dieses Kriterium wird aber nicht vollständig erfüllt, aber nahezu über die facettierte Suche bei der offiziellen Implementierung und über die verschiedenen Einstellungsmöglichkeiten bei meiner Implementierung. Ich schätze deshalb dies als ein kleines Usability-Problem ein.
5. Ist die anfängliche Sortierung der Treffer (Ranking) für den Anwender transparent?	Bei meiner Implementierung gibt es kein Ranking, sondern nur eine alphabetische Sortierung der Treffer nach dem Repositoriumsnamen. Bei der offiziellen Implementierung gibt es ein Ranking. Dies erschließt sich aber nicht sofort und es wird auch nicht kommuniziert, wie das Ranking zustande kommt. Für beide Seiten schätze ich dies als ein

	kleines Usability-Problem ein.
Präsentation & Zugriff - Manipulation der Trefferliste	
1. Ist es möglich, die Anzahl der Treffer pro Seite anzupassen, sofern die Trefferliste über mehrere Seiten paginiert ist?	Nein, das ist nicht möglich. Ich schätze dies als ein mittleres Usability-Problem ein, da es Einfluss auf die Übersichtlichkeit der Trefferliste hat.

Tabelle 3: Übereinstimmende Usability-Probleme der beiden Implementierungen

Wurden in Tabelle 3 die Usability-Probleme gezeigt, die bei beiden Implementierungen gleich sind, folgen in Tabelle 4 die Unterschiede:

Ausgewählte Kriterien	***Ergebnis der Bewertung***
Recherche im Bestand - Allgemein	
2. Stehen dem Anwender jederzeit angemessene Extraktionsmechanismen (z.B. Email, Print, Speichern, etc.) zur weiteren Verwertung seiner Recherchen und Treffer zur Verfügung?	Dieses Kriterium kann nur die offizielle Implementierung von re3data.org erfüllen. Meine Implementierung bietet keine angemessenen Extraktionsmöglichkeiten wie Drucken und Speichern an. Ich betrachte dies bei meiner Implementierung als ein schweres Usability-Problem, da der Nutzer seine Rechercheergebnisse schlecht „offline“ nutzen oder bspw. per E-Mail teilen kann.
Recherche im Bestand - Suchen & Erkunden	
2. Ist während einer Recherche jederzeit ein Wechsel zwischen den unterschiedlichen Suchstrategien möglich?	Bei der offiziellen Implementierung gibt es mit diesem Kriterium ein kleines Usability-Problem, was die Flexibilität bei der Suche einschränkt. Der Wechsel ist zwar möglich, jedoch werden bspw. beim Leeren oder Anpassen der Freitextsuche auch gleichzeitig die ausgewählten Kriterien der facettierten Suche verworfen. Diese müssen nach der Anpassung der Freitextsuche wieder einzeln ausgewählt werden. Meine Implementierung erfüllt dieses Kriterium vollständig.
Recherche im Bestand - Suchen & Erkunden - Erweiterte Suche	
3. Ist die Sortierung innerhalb von Drop-Down-Menüs intuitiv verständlich (alphabetisch, nach Relevanz)?	Bei meiner Implementierung folgt die Sortierung innerhalb der Dropdown-Menüs keinem Muster, was den Nutzer verwirren könnte. Er hat aber die Möglichkeit in den Dropdown-Menüs selbst nach einem Begriff zu suchen. Ich bewerte das Fehlen einer intuitiven Sortierung als ein mittleres Usability-Problem. Die offizielle Implementierung verwendet mit einer Ausnahme keine Dropdown-Menüs, deswegen gibt es dort kein Usability-Problem, da das eine Dropdown-Menü nur zwei Einträge hat.
Präsentation & Zugriff - Allgemein	
1. Unterstützt die Dialoggestaltung (z.B. die verwendeten Navigationselemente) einen einfachen Wechsel zwischen Suche, Trefferlisten und Detailanzeigen?	Bei meiner Implementierung gibt es ein mittleres Usability-Problem dadurch, dass die Detailansicht sich in der bestehenden Webseite öffnet und der Wechsel zwischen der Übersichtsseite und der Detailansicht nur über die Browserbuttons funktioniert. Dieses Feature muss noch verbessert werden und ist nicht besonders fehlertolerant. Die offizielle Implementierung dagegen löst das Problem durch Öffnen eines separaten Tabs für jede Detailansicht. Bei meiner Implementierung bewerte ich dieses unausgereifte Feature mit einem mittleren Usability-Problem.
Präsentation & Zugriff - Darstellung der Trefferliste	
6. Ist es transparent, in wie weit und weshalb die einzelnen Treffer der Suchanfrage entsprechen?	Bei meiner Implementierung werden alle Suchkriterien angezeigt, die zu der Trefferliste geführt haben und größtenteils als Labels farblich hervorgehoben. Bei der offiziellen Implementierung kann die ausgeklappte facettierte Suche sehr lang werden, darunter kann die Transparenz leiden. Außerdem wird in der Trefferliste nicht farblich angezeigt, welches Suchkriterium zu dem Treffer geführt hat. Deshalb bewerte ich dies bei der

	offiziellen Implementierung als ein mittleres Usability-Problem.
Präsentation & Zugriff - Darstellung der Detailansicht	
4. Ist es möglich, direkt zwischen den Detailansichten der einzelnen Treffer zu navigieren?	Dieses Kriterium wird nur von der offiziellen Implementierung brauchbar umgesetzt, dadurch, dass für jede Detailansicht ein einzelner Tab geöffnet wird. Bei meiner Implementierung ist dies dagegen nicht möglich. Deshalb bewerte ich dies als ein mittleres Usability-Problem.
6. Werden Extraktionsmechanismen wie Email, Bookmark und Drucken angeboten?	Extraktionsmechanismen wie von diesem Kriterium gefordert werden nur von der offiziellen Implementierung richtig unterstützt. Meine Implementierung bietet diese Möglichkeiten nicht, weshalb ich dies als ein schweres Usability-Problem betrachte, da es die Weiternutzung und Distribution der Suchergebnisse negativ beeinflusst.
Präsentation & Zugriff - Manipulation der Trefferliste	
2. Ist es möglich, die Trefferliste nach unterschiedlichen Kriterien zu sortieren?	Bei der offiziellen Implementierung kann der Nutzer die Trefferliste nach einem internen Ranking sortieren lassen oder alphabetisch. Bei meiner Implementierung besteht diese Möglichkeit nicht, hier gibt nur eine alphabetische Sortierung, weshalb ich dies als ein kleines Usability-Problem bei meiner Implementierung bewerte.

Tabelle 4: Unterschiedliche Bewertung der Kriterien bei beiden Implementierungen

3.1.1 Ergebnisse der Heuristischen Evaluation

Im Folgenden möchte ich die Probleme zusammenfassen, die durch die Heuristische Evaluation bei beiden Implementierungen hervorgetreten sind: Das erste Problem betrifft die Suche im Suchschlitz. Dort werden keine Suchoperatoren unterstützt und es ist auch nicht ersichtlich, wie die Begriffe im Suchschlitz miteinander verknüpft werden. Außerdem fehlt die Möglichkeit für den Nutzer auf eine Suchhistorie zu zugreifen, mit der Ausnahme, dass der eingesetzte Webbrowser die Formulardaten speichert. Dieses Feature wirkt sich aber nur auf den Suchschlitz aus. Generell gibt es bei den Implementierungen kaum assistierende Funktionen bei der Suche. Auch das Resultat der Suche, die Darstellung der Trefferliste, lässt bei beiden Implementierungen kaum Rückschlüsse auf das eingesetzte Ranking, wenn vorhanden, zu. Außerdem kann der Nutzer die Anzahl der Treffer pro Seite nicht selbst festlegen. Das System gibt immer eine feste Anzahl an Treffern aus.

Habe ich gerade die gemeinsamen Probleme beschrieben, möchte ich nun noch einen Blick auf die Unterschiede werfen: Ein großes Manko meiner Seite ist, dass sie keine angemessenen Extraktionsmechanismen der Suchergebnisse für die Weiternutzung anbietet. Weder lässt sich ein Bookmark setzen, noch kann die Seite komplett ausgedruckt werden. Meine Implementierung hat insgesamt sechs Dropdown-Menüs, die zwar durchsuchbar sind, aber sonst keine Logik oder Hierarchie ihrer Inhalte besitzen. Außerdem ist der Wechsel von der Übersichtsseite zur Detailansicht bei meiner Seite nur rudimentär gelöst und nicht besonders fehlertolerant. Hat der Nutzer einmal die Detailansicht geöffnet, muss er wieder zurück zur Übersichtsseite wechseln, wenn er eine andere Detailansicht aufrufen möchte. Als letztes Problem, welches durch die Heuristische Evaluation ermittelt wurde, ist, dass die Trefferliste nicht nach unterschiedlichen Kriterien sortiert werden kann.

Bei der offiziellen Implementierung tritt dagegen der Fehler auf, dass alle ausgewählten Facetten verworfen werden, wenn der Suchschlitz in die Suche mit einbezogen wird.

Außerdem ist für die Nutzer nicht unmittelbar offensichtlich, welche Kriterien ausgewählt wurden und damit zu den Suchergebnissen geführt haben. Diese Informationen sind zwar vorhanden, aber der Nutzer muss sie sich auf der gesamten Seite zusammen suchen.
Damit endet die Darstellung der Heuristischen Evaluation der Implementierungen. Das darauffolgende Unterkapitel beinhaltet das Usability-Testing.

3.2 Vergleichender Usability-Test zweier Webservices

Ein Usability-Test ist die Methode der Wahl, wenn es darum geht ein fertiges oder ein in der Entwicklung befindliches (Software-) Produkt mit echten Testpersonen zu evaluieren. Die Literatur unterscheidet dabei zwischen induktiven Tests, die der formativen Evaluation dienen und deduktiven Tests, die bei der summativen Evaluation eingesetzt werden (Sarodnick and Brau 2011, Vgl. S. 163). „Sowohl die Heuristische Evaluation als auch empirische Methoden sind (…) für eine umfassende Beurteilung geeignet." (Sarodnick and Brau 2011, S. 204) Die „Summative Evaluation kann [aber] kaum mit der Heuristischen Evaluation erfolgen, hier sind Fragebogen oder Usability-Tests deutlich überlegen." „In frühen Phasen empfiehlt sich aber der Einsatz der Heuristischen Evaluation" (Sarodnick and Brau 2011, S. 203), wenn eine formative Evaluation durchgeführt werden soll. Können induktive Tests auch am realen Arbeitsort durchgeführt werden, empfiehlt es sich für deduktive Tests diese, in einem Labor durchzuführen, „da so standardisierte Testbedingungen geschaffen werden können, die für eine Vergleichbarkeit grundlegend sind." Beim Durchführen eines deduktiven Usability-Tests sollte in der Regel ein AB-Test durchgeführt werden: Die eine Versuchsgruppe testet dabei zuerst die Variante A des Systems und danach die Variante B. Die andere Versuchsgruppe testet in genau umgekehrter Reihenfolge. „Auf diese Weise werden Beeinflussungen durch die Reihenfolge der Präsentation ausgeschlossen." (Sarodnick and Brau 2011, S. 164) „Die Gruppe der Testpersonen [sollte] die Bandbreite der Endbenutzer angemessen widerspiegeln, also repräsentativ sein. Außerdem sollten Testpersonen das zu testende System nicht kennen, da sonst viele Probleme nicht mehr auftreten bzw. umgangen werden würden." (Sarodnick and Brau 2011, S. 167) Wie schon im Literaturbericht erwähnt, kann eine geringe Anzahl an Testpersonen einen großen Anteil der Usability-Probleme aufdecken. „Letztendlich hängt [aber] die Anzahl der sinnvollerweise einbezogenen Testpersonen von dem Einsatzfeld des Systems, der Zahl der Tests im Laufe des Entwicklungsprozesses, der Heterogenität der Zielgruppe und nicht zuletzt [von] dem Budget für die Entwicklung ab." (Sarodnick and Brau 2011, S. 167) Außerdem sollte sich das Test-Team für einen Ort entscheiden. Der Vorteil des stationären Labors ist, dass vorab weniger abgesprochen werden muss und dass die Technik in der Regel wie gewohnt funktioniert (Sarodnick and Brau 2011, Vgl. S. 168 - 169). „Mobile Lösungen haben [dagegen] den Vorteil, dass der Test im Feld, also in der realen Umgebung durchgeführt werden kann. So können häufig realistischere Bedingungen geschaffen werden." (Sarodnick and Brau 2011, S. 169) Hat sich das Test-Team für einen Ort entschieden, sollte eine geeignete Methode ausgewählt werden. Hier bietet sich neben der Logile-Analyse, die mit einem sehr hohen Vor- und Nachbereitungsaufwand verbunden ist, das laute Denken (Thinking Aloud) an, welches sehr hilfreiche qualitative Informationen

ergeben kann (Sarodnick and Brau 2011, Vgl. S. 170). Ein Problem mit der Methode des lauten Denkens kann sein, dass sie zu einer Doppelbelastung der Testperson führen kann und dadurch die Testgeschwindigkeit reduziert wird (Sarodnick and Brau 2011, Vgl. S. 171). Da ich aber keine Performance-Messungen in den Test integriert habe, habe ich dieses Problem nicht beachtet. Ein anderes Problem mit dieser Methode ist, dass sie zu einer Veränderung und Vergegenwärtigung von Problemlösungsstrategien führen kann und „dass [dabei] weniger oder andere Probleme auftauchen können als unter realen Bedingungen." „Es gibt [aber] auch Untersuchungen, die zeigen, dass Aufgaben mit lautem Denken schneller gelöst oder weniger Fehler gemacht werden (siehe z. B. Berry & Broadbent, 1990; Wright & Converse, 1992)." (Sarodnick and Brau 2011, S. 171) Ich habe mich bei meinem Usability-Test trotz dieser Einwände für die Methode des lauten Denkens entschieden und zusätzlich ein Bildschirmaufzeichnungsprogramm eingesetzt, das alle Aktivitäten des Nutzers auf dem Bildschirm aufzeichnet. Somit kann nach dem Usability-Test auf ein von Nielsen empfohlenes Videofeedback verzichtet werden. Bevor die beschriebenen Methoden angewendet werden können, sollte ein Usability-Testplan erstellt werden. Dieser ist ein unverzichtbarer Teil des Testprozesses. Der Testplan beschreibt, was gemacht werden soll und wie es gemacht werden soll. Er ist essentiell für das Usability-Team, indem er als Handlungsanweisung dient, auf die sich alle Beteiligten berufen können. Deshalb sollte jeder, von der Planungsabteilung über die Gruppenmitglieder, die den Test ausführen, bis zu den Personen, die die Ergebnisse analysieren und auswerten, Zugriff auf den Plan selbst und seine Versionsgeschichte haben. Ein Testplan sollte so geschrieben sein, dass ein Außenstehender diesen verstehen und an seine Bedürfnisse anpassen kann. Insbesondere dieser Aspekt ist für die Wiederverwendbarkeit, bspw. in einer internationalen Firma, wichtig (Pearrow 2007, Vgl. S. 240).

3.2.1 Der Usability-Testplan

Nach Pearrow umfasst ein Usability-Testplan insgesamt 11 Abschnitte, die hier im Folgenden erläutert werden sollen. Dabei werde ich so verfahren, dass der hier vorgestellte Testplan alle relevanten Informationen enthält, die in einem Usability-Testplan stehen sollten. Dabei kann es vorkommen, dass ich Informationen aus vorangehenden Kapiteln der Vollständigkeit halber nochmal kurz erwähne.

3.2.1.1 Abschnitt 1: Zweck der Seite

Um die Effizienz einer Webseite zu testen, müssen zunächst die Erwartungshaltungen an die Seite beschrieben werden und ihre Funktionen, die sie beabsichtigt zur Verfügung zu stellen (Pearrow 2007, Vgl. S. 241):
Die Funktionen der beiden re3data.org-Webserviceimplementierungen sind die Bereitstellung eines Dienstes, der dem Wissenschaftler, Bibliothekar oder Forschungsdatenmanager die Möglichkeit bietet, ein für seinen Kontext geeignetes Forschungsdatenrepositorium zu finden. Damit ist re3data.org ein wichtiger Baustein der wissenschaftlichen Informationsinfrastruktur und dem zu ihr gehörenden wachsenden Bereich des digitalen Forschungsdatenmanagements. re3data.org soll eine Schlüsselrolle

beim Finden eines Forschungsdatenrepositoriums spielen. Dieses Finden wird durch die Verwendung von aktuellen Webtechnologien wie PHP, Javascript, HTML und CSS befördert und unterstützt. Der Usability-Test soll dabei ans Licht bringen, inwieweit die entsprechende Zielgruppe ihr Ziel effektiv, effizient und zufriedenstellend erreichen kann. Erst wenn ein Webservice „usable“ ist, kann er auch sein volles Potential entfalten. Der Auftraggeber des Usability-Tests möchte durch die vergleichende Usability-Evaluation der beiden Dienste in Erfahrung bringen, welcher Dienst die bessere Usability aufweist und damit für seine Zielgruppe besser nutzbar ist.

3.2.1.2 Abschnitt 2: Problemstatements

Problemstatements sind kurze Sätze, die zusammenfassen, welche Art von Fragen während des Tests beantwortet werden sollen. Problemstatements verwenden eine konkretere Sprache als in den abstrakten Beschreibungen aus Abschnitt 1 und beginnen entsprechende Unteraufgaben der Seite zu definieren, die erzielt werden müssen, damit der Zweck der Seite als erfüllt betrachtet werden kann (Pearrow 2007, Vgl. S. 242).
Problemstatements für meine Evaluation sind:

1. Können die Nutzer die Bedienelemente identifizieren, mit denen sie ihre Suchkriterien auswählen können?
2. Wissen die Nutzer, wie sie detaillierte Informationen über ein spezifisches Repositorium abfragen können?
3. Können die Nutzer die verschiedenen Sucheinstiege nutzen und verstehen sie deren Bedeutung für den Suchprozess?

3.2.1.3 Abschnitt 3: Benutzerprofile

Usability-Tests erfordern eine kleine Teilmenge der Grundgesamtheit der Nutzer. Diese Teilmenge sollte im Idealfall durch das Ziehen einer Zufallsstichprobe entstehen (Pearrow 2007, Vgl. S. 242).
Enthält die Zufallsstichprobe die für den Test notwendigen Benutzerprofile der Versuchspersonen, also in meinem Fall Bibliothekare, Wissenschaftler oder Forschungsdatenmanager, kann am Ende des Tests mit statistischen Methoden auch auf das Testverhalten der Grundgesamtheit, die auch aus den eben erwähnten Personengruppen bestehen sollte, geschlossen werden. Bei der Auswahl der Testpersonen sollte deshalb darauf geachtet werden, dass sie repräsentativ für die spätere Nutzergruppe sind (Sarodnick and Brau 2011, Vgl. S. 237). Da ich bei meinem Usability-Test keine Zufallsstichprobe aus der Grundgesamtheit der potentiellen Nutzer gezogen habe, kann ich folglich auch keine allgemeingültigen Aussagen über diese treffen. Hielte sich das Test-Team dagegen an diese Vorgaben, wäre dies aber möglich.

3.2.1.4 Abschnitt 4: Methodologie

Hier wird beschrieben, wie der Usability-Test abläuft. Zu Beginn des Tests sollte eine kurze Einführung stattfinden, die den Testablauf beschreibt und auch die Ziele des Tests erläutert (Sarodnick and Brau 2011, Vgl. S. 240).

Die Webservices liefen während des Tests auf meinem Notebook, das von den Testpersonen mit der Tastatur und der Maus bedient wurde. Als Testpersonen wurden von mir zwei Informationswissenschaftlerinnen ausgewählt. Der Test fand während eines Werktages statt. Neben mir und der Testperson waren keine weiteren Personen in dem Raum als der Test aufgezeichnet wurde. Der zweite Test wurde auch allein mit der zweiten Versuchsperson durchgeführt. Neben dem Bildschirminhalt wurden auch die Kommentare der Testpersonen über das Mikrofon des Notebooks von dem Bildschirmaufzeichnungsprogramm aufgezeichnet. Als Methode während des Tests wurde das laute Denken eingesetzt. Ich saß während des Tests neben der Versuchsperson und habe schriftliche Notizen über den Versuchsablauf angefertigt. Vor dem Test händigte ich der Versuchsperson ein Blatt mit sechs Standardaufgaben pro Webseite aus und bat sie dieses durchzulesen und ggf. Fragen zu stellen und Verständnisprobleme zur Sprache zu bringen. Nachdem der Usability-Test pro Testperson weniger als eine Stunde gedauert hatte, habe ich noch mit der Testperson ein abschließendes Gespräch über den Test geführt.

3.2.1.5 Abschnitt 5: Testumgebung und verwendete Geräte

Die Testumgebung war ein Raum im Institut für Bibliotheks- und Informationswissenschaft der Humboldt-Universität zu Berlin. Das verwendete Notebook ist ein Thinkpad der Firma Lenovo gewesen, auf dem das Betriebssystem Windows 10 läuft. Die Bildschirmaufzeichnungssoftware war der Apowersoft Screenrecorder. Der verwendete Webbrowser war Google Chrome und der Webserver auf dem meine Implementierung lief war Apache Tomcat.
Falls ein Equipment nicht ausgetauscht werden sollte für nachfolgende Tests, sollte dies hier erwähnt werden (Pearrow 2007, Vgl. S. 244). In meinem Fall ist dies nicht gegeben.

3.2.1.6 Abschnitt 6: Das Test-Team

Die minimale Anzahl von Mitarbeitern bei einem Usability-Test sollte zwei sein, es sei denn, es wird automatisierte Software eingesetzt (Pearrow 2007, Vgl. S. 244), wie in meinem Fall. Während eines Usability-Tests sind unterschiedliche Rollen zu besetzen, die im Folgenden kurz erläutert werden. Der Moderator interagiert mit den Versuchspersonen, moderiert die Sitzung, liest die Skripte und achtet darauf, dass der Test weiter voran schreitet. Der Data Logger notiert die Aktionen der Teilnehmer und ihre Fehler und alle relevanten Beobachtungen. In meinem Fall habe ich diese beiden Rollen ausgefüllt. Der Timer misst die Zeit, die ein Teilnehmer für das Beenden einer Aufgabe braucht und macht sich bemerkbar, wenn die Zeit um ist. Diese Rolle ist nur für die Performance-Messungen notwendig. Der Videooperator steuert die Kamera(s) im Versuchsraum und ist für die Aufnahme-Technik verantwortlich. Der Webseitenspezialist bringt Expertenwissen mit, um zu entscheiden, ob eine Aufgabe vom Teilnehmer gelöst wurde. Es ist wichtig im Testplan zu erwähnen, wer zum Team gehörte und welche Rollen und Aufgaben er oder sie dort ausfüllte (Pearrow 2007, Vgl. S. 245).

3.2.1.7 Abschnitt 7: Evaluationsmaße

Dieser Abschnitt erläutert die Kriterien, die entscheiden, ob eine Aufgabe vom Teilnehmer gelöst wurde oder nicht. Gängige Maße, ob eine Aufgabe gelöst wurde, können die Anzahl der Klicks sein, die ein Nutzer gebraucht hat oder die verstrichene Zeit zum Lösen einer Aufgabe (Pearrow 2007, Vgl. S. 245). Für meinen Usability-Test habe ich keine Evaluationsmaße direkt definiert. Eine Aufgabe galt als gelöst, wenn die Testperson alle gesuchten Eigenschaften eines Repositoriums, die in den Standardaufgaben abgefragt wurden, gefunden hat.

3.2.1.8 Abschnitt 8: Die Liste der Standardaufgaben

In diesem Abschnitt werden die individuellen Aufgaben definiert, die von den Problemstatements aus Abschnitt 2 abgeleitet wurden, um diese zu beantworten. In der Regel hat jede Aufgabe eine Textbeschreibung, eine Bedingung für das erfolgreiche Absolvieren und den Startzustand, ab dem diese Aufgabe gelöst werden soll (Pearrow 2007, Vgl. S. 245 - 246). „Die Aufgaben müssen im Inhalt und in der Bandbreite repräsentativ sein für den Einsatzbereich." (Sarodnick and Brau 2011, S. 236) Außerdem sollten sie, wenn möglich aus der Praxis entnommen sein und alle relevanten Bereiche des Systems abdecken. Die Aufgaben sollten in schriftlicher Form während des Tests vorliegen (Sarodnick and Brau 2011, Vgl. S. 236 - 237). Im Anschluss an die Aufgabenbearbeitung kann optional noch ein Fragebogen von der Testperson ausgefüllt werden, um bspw. soziodemographische Daten oder Vorerfahrungen ermitteln zu können (Sarodnick and Brau 2011, Vgl. S. 243). Bevor die Standardaufgaben in der Praxis eingesetzt werden, sollte noch ein Pretest durchgeführt werden, um etwaige Probleme bei der Bearbeitung zu erkennen und die Standardaufgaben dementsprechend anzupassen bzw. zu korrigieren. Einen Pretest habe ich bei der Vorbereitung meiner Usability-Evaluation nicht durchgeführt. Es folgt in der Tabelle 5 meine Liste der Standardaufgaben und ihre Begründung:

Standardaufgabe	***Begründung***
1. Suche nach einem Repositorium, das Forschungsdaten aus den Fachgebieten [„Mathematik" / „Biologie"] und (logisch) [„Statistik" / „Neurowissenschaften"] enthält.	Dies ist eine Aufgabe, die bei beiden Implementierungen ähnlich leicht zu lösen ist. Die Lösung führt über die Facette „Subjects" oder über ein entsprechendes Dropdown-Menü.
2. Suche ein Repositorium, das dem Forscher erlaubt seine Forschungsdaten [nach einer Registrierung / ohne Hindernisse] hochzuladen. a. Wenn Du eins gefunden hast, schaue nach, für welche [„Subjects" (Fachgebiete) / „Content-Types" (Datentypen)] dieses Repositorium ausgelegt ist und wähle [ein / einen] für Dich relevante[s / n] aus.	Diese Aufgabe soll testen, ob die Testperson die tiefer liegenden Suchkriterien findet und entsprechend die richtigen auswählen kann. Bei der offiziellen Implementierung führt der Weg über eine Facette und bei meiner Implementierung über die Sidebar. Außerdem wird untersucht, wie die Testperson ein relevantes „Subject" auswählt: entweder über ein Label, eine Facette oder über ein Dropdown-Menü.
3. Suche jetzt ein Repositorium, das nach den Open Access-Grundsätzen von re3data.org betrieben wird und (logisch) das seine Forschungsdaten mit	Diese Aufgabe soll herausfinden, ob das Ranking bei der offiziellen Implementierung und das damit zusammenhängende Icon-System verstanden wird: Je mehr Icon-Kriterien ein Repositorium erfüllt, desto höher wird es gerankt. Das Open Access-Icon führt automatisch

[Autorenidentifikatoren verknüpft / mit Persistent Identifiern versieht]. a. Schaue in dem von Dir gefundenen Repositorium nach, welche Data License (Datenlizenz) (bspw. CC oder Public Domain) es unterstützt.	zu einem höheren Ranking. Bei meiner Implementierung gibt es dagegen eine Checkbox, um das Open Access-Kriterium auszuwählen. Weiterhin bietet sich bei meiner Implementierung eine Checkbox an, den zweiten Teil des ersten Kriteriums zu lösen, bei der offiziellen Implementierung kann dieses Kriterium als Facette ausgewählt werden. Der zweite Teil der Suche soll untersuchen, ob die Testperson die Detailansicht auswählen kann und dort den richtigen Reiter für die Datenlizenz findet.
4. Suche ein Repositorium, das als Data License [„CC0“ / „Copyright“] hat.	Diese Aufgabe lässt sich bei der offiziellen Implementierung wieder sehr einfach über eine Facette lösen. Ob dieser Lösungsweg gefunden wird, soll untersucht werden. Bei meiner Implementierung führt diese Suchanfrage zu Problemen, da die Testperson dieses Kriterium nicht direkt auswählen kann.
5. Suche ein Repositorium, das als [„ServiceProvider“ / „DataProvider“] arbeitet. a. Überprüfe, ob das Repositorium auch ein [„DataProvider“ / „ServiceProvider“] ist.	Hier soll überprüft werden, ob bei der offiziellen Implementierung die Facette gefunden wird und ob bei meiner Implementierung der Sidebar mit seinen Checkboxen dafür verwendet wird.
6. Suche am Ende der 6 Aufgaben ein Repositorium, das drei beliebige Keywords von Fachgebieten enthält und als verantwortliche (responsible) Institution die [„National Institutes of Health“ („NIH“) / „NASA“] hat. Entscheide selbst, ob Du von Null beginnst, also alle Filter zurücksetzt oder mit den bisher ausgewählten Suchkriterien weitersuchst.	Diese Aufgabe lässt unterschiedliche Lösungsstrategien bei der offiziellen Implementierung zu: über der Suchschlitz oder über eine Facette. Dagegen ist der zweite Teil der Aufgabe mit der offiziellen Implementierung nur über den Suchschlitz zu lösen und bei meiner Implementierung über das Dropdown-Menü. Es wird interessant zu sehen, welche Lösungswege von den Testpersonen eingeschlagen werden.

Tabelle 5: Standardaufgaben und ihre Begründung

3.2.1.9 Abschnitt 9: Die Resultate

In diesem Abschnitt wird für jede Aufgabe der Erfolg oder Misserfolg pro Teilnehmer bzw. Testperson beschrieben, die Gesamtzeit zur Lösung der Aufgabe und die Anzahl der Klicks pro Aufgabe, wenn das Test-Team auch Performance-Messungen machen will (Pearrow 2007, Vgl. S. 246). Meinungen, Spekulationen, Hypothesen und alles, was nicht objektive Daten sind, gehören nicht in diese Abschnitt, sondern in die beiden Folgenden. In der Regel sollte die Auswertung der Testergebnisse keine Rückschlüsse auf die Testpersonen zulassen. Anonymität ist hier entscheidend (Sarodnick and Brau 2011, Vgl. S. 241): sowohl aus Datenschutzgründen als auch aus dem Grund, Geschäftsgeheimnisse zu bewahren. Als Nächstes möchte ich die Ergebnisse des Usability-Tests von Testperson A und Testperson B aufzeigen:

Offizielle Webseite		
Frage	***Auswertung Testperson A***	***Auswertung Testperson B***
1. Suche nach einem Repositorium, das Forschungsdaten aus den Fachgebieten „Mathematik“ und (logisch) „Statistik“ enthält.	Testperson A verwendet die Facette „Subjects“ wird dort aber von der Anzahl der Auswahlmöglichkeiten überwältigt. Testperson A wünscht sich eine Ebenenstruktur. „Statistics“ wird nicht gefunden. Testperson A nutzt daraufhin die Browsersuche. Das entsprechende	Die Testperson B verwendet weder die Facette noch den Suchschlitz, sondern verwendet die Funktion „Browse by Subject“. Dort wählt sie zunächst „Mathematics“ aus und findet aber durch die automatische

	Subject wird nicht gefunden, da es kein Repositorium gibt, das beide erfüllt. Die automatische Eingrenzung wird nicht bemerkt und nicht als hilfreich wahrgenommen.	Einschränkung nicht das Fachgebiet „Statistics“. Deshalb wählt die Testperson B noch ein Fachgebiet aus, in dem Statistik vorkommt „Statistical Physics“.
2. Suche ein Repositorium, das dem Forscher erlaubt seine Forschungsdaten nach einer Registrierung hochzuladen. a. Wenn Du eins gefunden hast, schaue nach, für welche „Subjects“ (Fachgebiete) dieses Repositorium ausgelegt ist und wähle ein für Dich relevantes aus.	Testperson A geht zunächst auf die Facette „dataUpload“ und wählt dort „restricted“ aus. Danach wählt sie außerdem noch bei der Facette „dataUploadRestrictions“ „registration“ aus. a.) Testperson A geht davon aus, dass bei der Facette „Subjects“ jetzt nur noch die angezeigt werden, die den oben ausgewählten Kriterien entsprechen. Testperson A wählt daraufhin ein beliebiges Subject aus. Dabei klickt sie aber nicht auf ein Label auf der Übersichtsseite, was auch möglich gewesen wäre. Testperson A wählt dann wieder ein beliebiges Repositorium für die Detailansicht aus. Testperson A bemängelt die Schwierigkeit des Verständnisses der re3data-Properties für Außenstehende.	Die Testperson B setzt den Filter zurück und navigiert zur „Data Upload“-Facette und danach zu „Data Upload Restrictions“-Facette und wählt dort „registration“ aus. a.) Die Testperson B öffnet die Detailansicht und wählt in der Detailansicht ein Subject über ein Label aus.
3. Suche jetzt ein Repositorium, das nach den Open Access-Grundsätzen von re3data.org betrieben wird und (logisch) das seine Forschungsdaten mit Autorenidentifikatoren verknüpft. a. Schaue in dem von Dir gefundenen Repositorium nach, welche Data License (Datenlizenz) (bspw. CC oder Public Domain) es unterstützt.	Testperson A klickt wieder als Erstes auf den „Reset-Filter“. Testperson A wählt dann den „Data Access“ und „Database Access“ als „Open“ aus. Daraufhin sucht sie nach der Facette für Autorenidentifikatoren. Die Testperson A weiß nicht, wo sie nach diesem Kriterium suchen. Sie geht als Erstes auf die Facette „PID Systems“, wird aber von ihr wieder verworfen. Testperson A übersieht die Facette „AID Systems“. Stattdessen gibt sie im Suchschlitz „ORCID“ ein und findet daraufhin ein geeignetes Repositorium. Vor der Suche öffnet sie ein neues Tab der Webseite, um dort dann im Suchschlitz die Kriterien einzugeben. Testperson A sucht außerdem noch nach „Author Claim“ im Suchschlitz. a.) Testperson A sucht im entsprechenden Tab der Detailansicht nach der Data License, die das Repositorium unterstützt.	Die Testperson B setzt den Filter zurück und wählt bei „Database Access“ „Open“ aus und wählt bei „Data Access“ ebenfalls „open“ aus. Danach wählt die Testperson B die Facette „AID System“ aus und entscheidet sich dort für „ORCID“. a.) Danach wechselt sie in die Detailansicht eines Repositoriums, um die gefundenen Kriterien zu überprüfen.
4. Suche ein Repositorium, das als Data License „CC0“ hat.	Testperson A setzt zunächst alle Suchkriterien zurück und geht dann auf die Facette „Data Licenses“ und wählt dort „CC0“ aus.	Die Testperson B setzt den Filter zurück und wählt die Facette „Data Licenses“ aus und wählt dort das Kriterium CC0 aus.

5. Suche ein Repositorium, das als „ServiceProvider“ arbeitet. a. Überprüfe, ob das Repositorium auch ein „DataProvider“ ist.	Testperson A setzt zunächst alle Suchkriterien zurück. Danach sucht die Testperson A nach der richtigen Facette und wählt dann die Facette „Provider types“ aus. Dort wählt sie beide möglichen Kriterien aus, um der Suchanfrage gerecht zu werden. Testperson A bemängelt am System, dass nicht sehr komfortabel angezeigt wird, welche Kriterien der Nutzer im Moment ausgewählt hat.	Die Testperson B setzt den Filter zurück und wählt die Facette „Provider types“ aus, um dort die Facette „serviceProvider“ aus ,um dann über das Öffnen eines Tabs in die Detailansicht eines beliebigen Repositoriums zu springen, um dort die gesuchten Kriterien zu überprüfen. a.) Zunächst findet die Testperson B nicht den richtigen Tab, auf dessen Seite die gewünschten Informationen stehen. Sie verwendet zur Suche auf einer Tabseite die Browsersuche.
6. Suche am Ende der 6 Aufgaben ein Repositorium, das drei beliebige Keywords von Fachgebieten enthält und als verantwortliche (responsible) Institution die „National Institutes of Health“ („NIH“) hat. Entscheide selbst, ob Du von Null beginnst, also alle Filter zurücksetzt oder mit den bisher ausgewählten Suchkriterien weitersuchst.	Testperson A setzt zunächst alle Suchkriterien zurück. Danach wählt die Testperson A die Facette „keywords“ aus und wählt dort Keywords aus, die mit dem Gesundheitsbereich zusammenhängen. Dies macht sie, da sie auch noch nach den NIH als verantwortliche Institution suchen soll. Die Testperson A stellt fest, dass die angebotenen Keywords sich verändern, je nachdem, welches Keyword man als Erstes auswählt. Um nach den NIH zu suchen, gibt die Testperson A im Suchschlitz NIH und danach „National Institutes of Health“ ein. Daraufhin werden die Keywords vom System wieder verworfen. Danach wählt die Testperson A nochmal drei Keywords aus, die zum NIH-Kontext gehören. Abschließend öffnet die Testperson A die Detailansicht und überprüft, ob die NIH als Institution angezeigt werden.	Die Testperson B setzt den Filter zurück und verwendet den Suchschlitz um nach den „National Institutes of Health“ zu suchen. Die Testperson B findet entsprechende Repositorien und wählt dann von einem Repositorium die Detailansicht aus. Danach verwendet sie die „Keywords“-Facette, um nach drei beliebigen Keywords zu filtern. Zum Abschluss wählt die Testperson B ein beliebiges Repositorium aus, um zu überprüfen, ob es die gewählten Keywords enthält.
Meine Implementierung		
Frage	***Auswertung Testperson A***	***Auswertung Testperson B***
1. Suche nach einem Repositorium, das Forschungsdaten aus den Fachgebieten „Biologie“ und (logisch) „Neurowissenschaften“ enthält.	Testperson A verwendet das Dropdown-Menü, um die Subjects auszuwählen. Nachdem die beiden Subjects ausgewählt wurden, versucht die Testperson A über das Öffnen eines neuen Tabs die Detailansicht zu öffnen, was misslingt. Daraufhin öffnet sie die Detailansicht direkt auf der Seite.	Die Testperson B verwendet das Dropdown-Menü und verwendet auch dessen Suchmöglichkeit, um die entsprechenden Subjects auszuwählen.
2. Suche ein Repositorium, das dem Forscher erlaubt seine Forschungsdaten ohne Hindernisse hochzuladen. a. Wenn Du eins	Bevor sie mit den neuen Kriterien sucht, löscht die Testperson A die Filter durch Klicken auf das Kreuz der Labels. Die Testperson A wählt dabei bei der linken Seite „Open“ für den	Die Testperson B setzt die beiden Filter durch Klicken auf die Labels zurück. Danach wählt sie die Checkbox „Open“ in der Sidebar aus, die zur Facette „Data Upload Type“ gehört.

gefunden hast, schaue nach, für welche „Content-Types“ (Datentypen) dieses Repositorium ausgelegt ist und wähle einen für Dich relevanten aus.	Data Upload Type aus, ist aber verwirrt, dass die Detailansicht weiter offen ist. Die Testperson A bemerkt nicht, dass die Detailansicht noch zu den alten Suchkriterien gehört und versucht außerdem durch Klicken auf ein Label in der Detailansicht einen Content-Type auszuwählen, was bei der aktuellen Implementierung nicht funktioniert. Dass das Auswählen nicht funktioniert, bemerkt die Testperson A nicht. Dann wählt sie einen Content-Type über das Dropdown-Menü oben aus und bemerkt jetzt, dass sie immer noch in der Detailansicht ist. Daraufhin setzt sie über den Button „Reset All“ alles zurück und beginnt die Kriterien für diese Aufgabe nochmal neu zu setzen.	a.) Danach wählt die Testperson B über ein graues Label auf der Übersichtsseite einen Contenttype aus.
3. Suche jetzt ein Repositorium, das nach den Open Access-Grundsätzen von re3data.org betrieben wird und (logisch) das seine Forschungsdaten mit Persistent Identifiern versieht. a. Schaue in dem von Dir gefundenen Repositorium nach, welche Data License (Datenlizenz) (bspw. CC oder Public Domain) es unterstützt.	Die Testperson A geht wieder zunächst auf „Reset All“. Danach wählt die Testperson A für den „Data Access“ „Open“ aus, revidiert diese Entscheidung aber, als sie die Checkbox für Open Access sieht. Außerdem wählt sie auch noch die Checkbox für „Persistent Identifier“ aus, um alle geforderten Suchkriterien abzudecken. a.) Die Testperson A öffnet die Detailansicht und klickt auf den richtigen Tab in dieser, um die Data License zu ermitteln.	Die Testperson B setzt den Filter zurück und wählt bei „Data Access“ „open“ aus. Danach setzt sie auch den Haken bei der „Open Access“-Checkbox. Die Testperson B findet es verwirrend, dass es unterschiedliche Access-Kriterien zur Auswahl gibt. a.) Danach wechselt sie in die Detailansicht und schaut nach, welche Datalicense das Repositorium unterstützt.
4. Suche ein Repositorium, das als Data License „Copyright“ hat.	Die Testperson A geht wieder zunächst auf „Reset All“. Testperson A bemerkt, dass sie nicht nach der Data License filtern kann - ein Schwachpunkt dieser Implementierung. Die Testperson A verwendet nun den Suchschlitz, um nach „Copyrights“ zu suchen, was misslingt. Die Testperson A wechselt daraufhin nach Absprache zur nächsten Frage.	Die Testperson B stellt fest, dass es keine Auswahlmöglichkeit gibt, um nach einer bestimmten Data License zu suchen.
5. Suche ein Repositorium, das als „DataProvider“ arbeitet. a. Überprüfe, ob das Repositorium auch ein „ServiceProvider“ ist.	Die Testperson A geht wieder zunächst auf „Reset All“. Danach verwendet sie die beiden Checkboxen zum „Provider Type“.	Die Testperson B setzt den Filter zurück und wählt in der Sidebar die Checkbox „DataProvider“ aus. Danach wechselt sie in die Detailansicht eines Repositoriums und überprüft, ob das Repositorium auch ein „ServiceProvider“ ist. Die Logik, dass der Nutzer auf den Back-Button klicken muss, um

		wieder auf die Übersichtsseite zu kommen, funktioniert in dieser Situation und scheint intuitiv zu sein. a.) Die Testperson B wählt dann beide Checkboxen in der Sidebar für den Providertype aus und wechselt wieder in die Detailansicht, um zu überprüfen, ob das Repositorium beide Providertypes anbietet.
6. Suche am Ende der 6 Aufgaben ein Repositorium, das drei beliebige Keywords von Fachgebieten enthält und als verantwortliche (responsible) Institution die „NASA“ hat. Entscheide selbst, ob Du von Null beginnst, also alle Filter zurücksetzt oder mit den bisher ausgewählten Suchkriterien weitersuchst.	Die Testperson A entfernt die Haken bei den Provider Types und gibt im Suchschlitz, bei dem man nur nach Subjects und Schlüsselwörtern suchen kann, die „NASA“ ein. Das Ergebnis sind keine gefundenen Repositorien. Daraufhin entdeckt sie das entsprechende Dropdown-Menü, vergisst aber den Suchschlitz zurückzusetzen, was sie kurz danach macht und wählt dann „NASA“ über das Dropdown-Menü aus. Die Testperson A wechselt dann noch einmal in die Detailansicht und schaut sich die Keywords eines Repositoriums an.	Die Testperson B setzt den Filter zurück und wählt im Dropdown-Menü zuerst die „NASA“ aus. Danach verwendet sie den Suchschlitz, um die drei beliebigen Keywords einzugeben. Dabei weiß die Testperson B nicht, wie die Keywords im Suchschlitz voneinander getrennt werden sollen. Zunächst verwendet sie ein Komma zur Trennung, was nicht funktioniert, bis sie ein Leerzeichen zur Trennung einsetzt. Danach gibt die Testperson B immer nur ein Keyword ein, um nach diesem zu suchen. In diesem Fall erscheint ein Repositorium bei der Suche nach jeweils einem unterschiedlichen Keyword oben auf der Übersichtsseite. Daraus und nach der Überprüfung der Detailansicht schließt die Testperson B, dass das Repositorium alle drei Keywords umfasst.

Tabelle 6: Die Resultate der Usability-Tests der beiden Testpersonen

3.2.1.10 Abschnitt 10: Die Diskussion

Das Ziel der Diskussion ist es, einen Sinn für die Resultate zu bekommen. Das bedeutet die gewonnenen Daten zu interpretieren und Erklärungen anzubieten, warum Dinge im Test so zustande gekommen sind (Pearrow 2007, Vgl. S. 247). Zunächst möchte ich die Usability-Probleme der offiziellen Webseite beschreiben, die ich durch den Usability-Test entdeckt habe: Ein Feature dieser Seite ist, dass es eine automatische Eingrenzung der Suchkriterien gibt. Das bedeutet, dass, wenn ein Suchkriterium ausgewählt wurde, die restlichen zur Verfügung stehenden Suchkriterien davon abhängen. Es kann also über die Facettensuche immer nur das Kriterium ausgewählt werden, was in Abhängigkeit der schon ausgewählten Kriterien zur Verfügung steht. Diese Logik war für eine Testperson nicht immer gleich ersichtlich und wurde erst nach fortschreitender Nutzungsdauer erkannt. Ein weiteres Problem dieser Implementierung ist, dass die Überschriften der Suchfacetten die re3data.org-Propertynamen sind und es Außenstehenden schwer fallen kann, ihre Bedeutung zu erkennen. Bspw. übersah eine Testperson die Facette „AID Systems“ als sie

nach Autorenidentifikatoren suchte. Ferner bemängelte eine Testperson, dass nicht transparent genug aufgezeigt wird, welche Kriterien der Nutzer im Moment ausgewählt hat. Der letzte zu kritisierende Punkt bei dieser Version ist, dass bei einem Wechsel der Suchstrategie, wenn der Nutzer von der Facettensuche zum Suchschlitz wechselt, die ausgewählten Facetten verworfen werden, wenn die Suche im Suchschlitz gestartet wird. Diesen Missstand habe ich, wie weiter oben beschrieben, auch schon bei der Heuristischen Evaluation entdeckt. Nachdem ich die gefundenen Usability-Probleme der offiziellen Webseite dargestellt habe, möchte ich jetzt die gefundenen Probleme meiner Implementierung aufzeigen: Das erste Problem ist, dass die Detailansicht nicht in einem separaten Browser-Tab angezeigt werden kann. Eine Testperson versuchte über das Kontextmenü des Browsers durch Klicken auf den Repositoriennamen einen Tab zu öffnen, was misslang. Außerdem ist der Wechsel zwischen Detailansicht und Übersichtsseite nicht klar erkennbar. Bspw. hielt eine Testperson die geöffnete Detailansicht für ein aktuelles Suchergebnis, weil sie über die Einstellungsmöglichkeiten die Kriterien weiter angepasst hatte. Die Testperson bemerkte nicht, dass die geöffnete Detailansicht zu einem früheren Suchergebnis gehörte. Ein weiteres Problem mit der Detailansicht ist, dass die dort angebotenen, z.T. farbigen, Labels nicht wie auf der Übersichtsseite auswählbar sind. Eine Testperson klickte auf ein Label in der Detailansicht und war verwundert, dass daraus keine Veränderung resultierte. Die andere Testperson fand es außerdem verwirrend, dass es unterschiedliche Access-Kriterien zur Auswahl gibt: einmal die Open Access-Checkbox und in der Sidebar eine Facette, über die sich ebenfalls Access-Kriterien auswählen lassen. Ein Problem, dass ich auch schon bei der Heuristischen Evaluation entdeckt hatte, wurde auch von einer Testperson erkannt. Es ist nämlich nicht klar ersichtlich, wie die Keywords im Suchschlitz voneinander getrennt werden sollen. Außerdem, so bemängelte eine Testperson, sei es nicht ersichtlich, welche Keywords ausgewählt und gefunden werden. Eine Testperson fand es auch verwirrend, dass sich bei meiner Implementierung die Suchmöglichkeiten sowohl links in der Sidebar als auch oben in der Mitte befinden. Die „ProviderType“-Facette auf meiner Seite wurde von einer Testperson auch als nicht besonders aussagekräftig bewertet. Ferner wurde das Dropdown-Menü für den API-Type als weniger wichtig wahrgenommen. Stattdessen sollte es eine Suchmöglichkeit nach häufiger gesuchten und verwendeten Suchkriterien geben. Kritisch wurde außerdem gesehen, dass bei den Dropdown-Menüs für die „responsible Institution“ nur die Akronyme angeboten werden. Damit komme ich zum letzten Abschnitt des Usability-Testplans. Dieser setzt im Folgenden den Fokus auf die aus den Resultaten abzuleitenden Empfehlungen.

3.2.1.11 Abschnitt 11: Die Empfehlungen für Veränderungen

In den Empfehlungen für Veränderungen sollten Implementierungen vorgeschlagen werden, um bestimmte entdeckte Usability-Probleme zu lösen. Dabei sollten die schwersten Usability-Probleme als Erstes erwähnt werden. Dies erlaubt es dem Auftraggeber, Veränderungen zu priorisieren und Ressourcen ggf. zu zuweisen, um die Probleme zu beseitigen (Pearrow 2007, Vgl. S. 247).

Bei meiner Implementierung sollte die Möglichkeit bestehen, Inhalte über den Browser angemessen zu extrahieren. Außerdem sollte darüber nachgedacht werden, die Suchkriterien an die URL anzuhängen, damit die Navigation innerhalb der Suchergebnisse besser mit dem Browser gelingt. Auch der Wechsel zwischen der Übersichtsseite und der Detailansicht sollte mit diesen erweiterten URLs besser gelingen. Ich könnte mir hier vorstellen, dass bei der Aus- und Abwahl eines Suchkriteriums die Übersichtsseite automatisch erscheint und mit den veränderten Kriterien neu geladen wird. Mit den Suchkriterien innerhalb einer aktuellen URL wäre auch ein Bookmark eines Suchergebnisses realisierbar, wenn die Programmlogik diese auswerten würde. Ferner wäre es wünschenswert, wenn ein Vor- und Zurückblättern innerhalb der Detailansicht möglich wäre, ohne dass der Nutzer zurück zur Übersichtsseite gehen muss. Das Aufrufen der Detailansicht in einem separaten Browser-Tab wäre außerdem von Vorteil für den Nutzer. Bei der offiziellen Implementierung wäre es dagegen wünschenswert, wenn die Facettennamen dahingehend angepasst werden würden, dass sie auch für Nutzer verständlich sind, die mit dem re3data.org-Metadatenschema nicht im Detail vertraut sind. Auch wäre eine bessere farbige Kennzeichnung der ausgewählten Kriterien an einer zentralen Stelle wünschenswert, um die Transparenz für den Nutzer zu erhöhen. Das Verwerfen von ausgewählten Kriterien, wenn der Nutzer den Suchschlitz verwendet, sollte möglichst auch vermieden werden.
Damit bin ich am Ende von Kapitel 3 und des Evaluationsteils meiner Arbeit. Es folgt ein kurzes Fazit und die Beantwortung der Forschungsfrage.

4 Fazit: Beantwortung der Forschungsfrage

Gerade durch den Evaluationsteil ist mir klar geworden, dass ich zwar ein weitgehend fertiges Softwareprodukt entwickelt habe, dieses aber im Detail noch nicht ausgereift ist. Durch einen neuen Release am Anfang des Jahres 2016 wurden viele Unzulänglichkeiten der alten offiziellen Webseite behoben. Dadurch sind die Metadaten von re3data.org um einiges besser durchsuchbar, auch wenn der Nutzer nicht die REST-Schnittstelle verwendet, um sich diese im Detail anzuschauen. Diese Veränderung der offiziellen Webseite hin zu einem Webservice, der für fast alle Eigenschaften eines erschlossenen Repositoriums eine facettierte Suche anbietet, zeigt, dass das Problem der Durchsuchbarkeit des alten Webservice nicht nur von mir als verbesserungswürdig betrachtet wurde. Der Entwickler des KIT ist damit einen Weg gegangen, den auch ich eingeschlagen habe, mit dem Ziel die Durchsuchbarkeit der re3data.org-Metadaten zu verbessern. Insofern bewerte ich dies als eine indirekte Bestätigung meines Projektes. Für mich war es eine spannende und interessante Aufgabe nur aus den öffentlich verfügbaren Metadaten mit einem Java-Programm eine Datenbank zu füllen, um diese durch eine Webseite bzw. Webanwendung zu erschließen und über eine Oberfläche bestimmte Metadaten abfragbar zu machen. Die Implementierung meines Triplestores orientiert sich sehr stark an der Struktur der XML-Dateien. Wünschenswert wäre hier noch, falls das Projekt für die Zukunft eine SPARQL-Schnittstelle für die re3data.org-Metadaten anbieten sollte, bspw. für jede Institution, auf die bestimmte Repositorien verweisen, nur eine Referenz anzubieten und nicht wie ich es

gemacht habe, für jedes Repositorium einen eigenen Metadatensatz der dazugehörigen Institution(en) zu erstellen. Dieser Ansatz würde aber bedeuten, dass schon die Ersteller der Metadaten sich auf ein kontrolliertes Vokabular für jede verantwortliche Institution einigen müssten und dass nicht, wie es die Erschließer wahrscheinlich immer noch machen, die Metadaten ohne Abgleich der bestehenden Metadaten eingetragen werden. Ein anderes Hindernis auf dem Weg zu Linked Data wäre, die Zahlenformate in den Metadaten konsistent zu halten. Bspw. hatte ich versucht, die Datumsangaben nicht vom Typ String sondern vom Typ Date im Triplestore zu speichern. Da es aber keine konsistenten Datumsangaben in den Metadaten gibt, konnte ich maximal nur das Jahr als Date-Typ eintragen, wenn ich keinen Abbruch meines Java-Programms zum Erstellen des Triplestores riskieren wollte. Dennoch funktioniert der Webservice bzw. die Webseite, die ich programmiert habe und könnte so, oder mit den Verbesserungsvorschlägen aus der Evaluation online gehen. Habe ich die Usability verbessert? Im Vergleich zu der alten Implementierung schon, die neue, aus diesem Jahr, hat aber nach meiner Einschätzung eine bessere Usability als mein Webservice. Trotzdem hoffe ich mit dieser Masterarbeit gezeigt zu haben, dass ein Projekt dieser Größenordnung durchaus realistisch von einer Person gestemmt werden kann und dass am Ende ein Produkt steht, das von Endanwendern genutzt werden kann, auch wenn es noch kleine Schwächen hinsichtlich der Effizienz und Effektivität des Produktes und der Zufriedenheit des Nutzers gibt.

5 Literaturverzeichnis

Bergsten, H. 2003. *JavaServer Pages*: O'Reilly Media.

Broschart, S. 2011. *Suchmaschinenoptimierung und Usability*: Franzis.

Butler, Keith A. 1996. "Usability engineering turns 10." *interactions* 3 (1):58-75.

BV, TIOBE Software. 2016. "TIOBE Index for January 2016." Accessed 26.01.2016. http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html.

Cervone, Frank. 2014. "Evidence-based practice and web usability assessment." *OCLC Systems & Services* 30 (1):11-14.

Çetin, Görkem, and M Göktürk. 2008. "A measurement based framework for assessment of usability-centricness of open source software projects." Signal Image Technology and Internet Based Systems, 2008. SITIS'08. IEEE International Conference on.

Delone, William H, and Ephraim R McLean. 2003. "The DeLone and McLean model of information systems success: a ten-year update." *Journal of management information systems* 19 (4):9-30.

Dhillon, B.S. 2004. *Engineering Usability: Fundamentals, Applications, Human Factors, and Human Error*: American Scientific Publishers.

Dillon, A. 2001. *Evaluation of Software Usability*. Edited by W. Karwowski, *International Encyclopedia of Ergonomics and Human Factors*: Taylor & Francis.

Doctrine. 2016. "5.5. One-To-Many, Bidirectional." Accessed 12.02.2016. https://doctrine-orm.readthedocs.org/projects/doctrine-orm/en/latest/reference/association-mapping.html#one-to-many-bidirectional.

Dr.Nic. 2013. "Composite Primary Keys." Accessed 10.03.2016. http://compositekeys.rubyforge.org/.

DuCharme, B. 2013. *Learning SPARQL*: O'Reilly Media.

Dunglas, K. 2013. *Persistence in PHP with the Doctrine ORM*: Packt Publishing.

Fischer, Oliver B. 2013. "Search and destroy." Accessed 08.02.2016. http://www.heise.de/developer/artikel/Volltextsuche-mit-ElasticSearch-1920454.html.

Fu, L.; Schmidt, K. 2001. *Usability Evaluation*. Edited by W. Karwowski, *International Encyclopedia of Ergonomics and Human Factors*: Taylor & Francis.

Fu, Limin, Gavriel Salvendy, and Lori Turley. 2002. "Effectiveness of user testing and heuristic evaluation as a function of performance classification." *Behaviour & Information Technology* 21 (2):137-143.

Gediga, Günther, and Kai-Christoph Hamborg. 2002. "Evaluation in der Software-Ergonomie: Methoden und Modelle im Software-Entwicklungsprozess." *Zeitschrift für Psychologie* 210 (1):40-57.

Goldhaber, Michael H. 1997. "The Attention Economy and the Net." Accessed 01.01.2016. http://firstmonday.org/ojs/index.php/fm/article/view/519/440.

Goll, J. 2014. *Architektur- und Entwurfsmuster der Softwaretechnik: Mit lauffähigen Beispielen in Java*: Springer Fachmedien Wiesbaden.

Hitzler, P., M. Krötzsch, S. Rudolph, and Y. Sure. 2007. *Semantic Web: Grundlagen*: Springer Berlin Heidelberg.

Hoffmann, Ralf, and Kirstin Krauss. 2004. "A critical evaluation of literature on visual aesthetics for the web." Proceedings of the 2004 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries.

Hornbæk, Kasper. 2006. "Current practice in measuring usability: Challenges to usability studies and research." *International journal of human-computer studies* 64 (2):79-102.

Jordan, Patrick W. 1998. "Human factors for pleasure in product use." *Applied ergonomics* 29 (1):25-33.

Larusdottir, Marta Kristin. 2011. "Usability evaluation in software development practice." In *Human-computer interaction–INTERACT 2011*, 430-433. Springer.

Lennard, Heike, and Melanie Surkau. 2011. *Benutzerevaluation und Usability-Test zur neuen Suchoberfläche Primo (Ex Libris)*.

Mahmood, Mo Adam, Janice M Burn, Leopoldo A Gemoets, and Carmen Jacquez. 2000. "Variables affecting information technology end-user satisfaction: a meta-analysis of the empirical literature." *International Journal of Human-Computer Studies* 52 (4):751-771.
Manhartsberger, M., and S. Musil. 2002. *Web usability: das Prinzip des Vertrauens*: Galileo Press.
McLaughlin, B., and J. Edelson. 2006. *Java and XML*: O'Reilly.
Miles, R., and K. Hamilton. 2006. *Learning UML 2.0*: O'Reilly Media.
MIT, ERCIM, Keio, Beihang. 2013. "SPARQL 1.1 Overview. W3C Recommendation 21 March 2013." Accessed 22.01.2016. https://www.w3.org/TR/sparql11-overview/.
Neumann, Alexander. 2016. "Java ist Programmiersprache des Jahres 2015 im TIOBE-Index." Accessed 26.01.2016. http://www.heise.de/newsticker/meldung/Java-ist-Programmiersprache-des-Jahres-2015-im-TIOBE-Index-3067171.html.
Nielsen, Jakob. 1994. *Usability Engineering*: AP Professional.
Nielsen, Jakob. 1995. "Why You Only Need to Test with 5 Users." Accessed 05.01.2016. https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/.
Niemeyer, P., and D. Leuck. 2013. *Learning Java*: O'Reilly Media.
Oscity, Patrick. 2015. "Wer sucht, der findet - Volltextsuche mit PostgreSQL und Elasticsearch." Accessed 08.02.2016. https://www.zweitag.de/de/blog/technologie/wer-sucht-der-findet-volltextsuche-mit-postgresql-und-elasticsearch.
Pearrow, M. 2007. *Web Usability Handbook*: Charles River Media.
Puscher, F. 2001. *Das Usability-Prinzip: Wege zur benutzerfreundlichen Website*: dpunkt-Verlag.
PYPL-Index. 2016. "PYPL PopularitY of Programming Language." Accessed 28.01.2016. https://pypl.github.io/PYPL.html.
Rahn, Johannes 2010. "Usability und User Experience." http://www.mathematik.uni-ulm.de/sai/ss10/guidesign/downloads/ux_vortrag.pdf.
Raza, Arif, Luiz Fernando Capretz, and Faheem Ahmed. 2012. "An open source usability maturity model (OS-UMM)." *Computers in Human Behavior* 28 (4):1109-1121.
RedMonk. 2015. "The RedMonk Programming Language Rankings: June 2015." Accessed 04.01.2016. https://redmonk.com/sogrady/2015/07/01/language-rankings-6-15/.
Reeves, Mark. 2015. "Top Ten Programming Languages, Which One Do you Prefer?" Accessed 04.01.2016. http://www.grataonline.com/top-ten-programming-languages-which-one-do-you-prefer/.
Richter, M., and M.D. Flückiger. 2013. *Usability Engineering kompakt: Benutzbare Produkte gezielt entwickeln*: Springer Berlin Heidelberg.
RoR-Documentation. 2016. "Active Record Basics." Accessed 04.03.2016. http://edgeguides.rubyonrails.org/active_record_basics.html.
Rücknagel, Jessika; Vierkant, Paul, et. al. 2015. Metadata Schema for the Description of Research Data Repositories. Accessed 05.01.2016. http://gfzpublic.gfz-potsdam.de/pubman/item/escidoc:1397899:6/component/escidoc:1398549/re3data_schema_documentation_v3_0.pdf.
Salunke, Smita, and Catherine Tuleu. 2015. "The STEP database through the end-users eyes—USABILITY STUDY." *International journal of pharmaceutics* 492 (1):316-331.
Sarodnick, F., and H. Brau. 2011. *Methoden der Usability Evaluation: wissenschaftliche Grundlagen und praktische Anwendung*: Huber.
Schenkman, Bo N, and Fredrik U Jönsson. 2000. "Aesthetics and preferences of web pages." *Behaviour & Information Technology* 19 (5):367-377.
Schirmbacher, Prof. Dr. Peter. 2014. 24.04.2014 VL Informationsinfrastrukturen Vorlesungsfolien.
SensioLabs. 2015. "Symfony: The Book." In. http://symfony.com/pdf/Symfony_book_master.pdf?v=4 (accessed September 25, 2015).

Sesame. 2016. "Sesame Webseite." Accessed 22.01.2016. http://rdf4j.org/about.docbook?view.
Sherman, P. 2006. *Usability Success Stories: How Organizations Improve by Making Easier-to-use Software and Web Sites*: Gower.
Symfony. 2016. "Databases and Doctrine." Accessed 12.02.2016. http://symfony.com/doc/current/book/doctrine.html.
Tate, B., and C. Hibbs. 2007. *Durchstarten mit Ruby on Rails*: O'Reilly.
Volentine, Rachel, Amber Owens, Carol Tenopir, and Mike Frame. 2015. "Usability Testing to Improve Research Data Services." *Qualitative & Quantitative Methods in Libraries*.
Wagner, Nicole, Khaled Hassanein, and Milena Head. 2014. "The impact of age on website usability." *Computers in Human Behavior* 37:270-282.
Weinhold, Thomas, Sonja Öttl, and Bernard Bekavac. 2011. "BibEval–Ein webbasierter Kriterienkatalog zur Usability-Evaluation von Bibliothekswebsites." *Bradfordizing als Re-Ranking-Ansatz in Literaturinformationssystemen*:10.
Whitefield, Andy, Frank Wilson, and John Dowell. 1991. "A framework for human factors evaluation." *Behaviour & Information Technology* 10 (1):65-79.
Wikipedia. 2016a. "Apache Tomcat." Accessed 22.01.2016. https://de.wikipedia.org/wiki/Apache_Tomcat.
Wikipedia. 2016b. "Dokumentenorientierte Datenbank." Accessed 20.02.2016. https://de.wikipedia.org/wiki/Dokumentenorientierte_Datenbank.
Wikipedia. 2016c. "Extensible Markup Language." Accessed 22.01.2016. https://de.wikipedia.org/wiki/Extensible_Markup_Language.
Wikipedia. 2016d. "JavaServer Pages." Accessed 03.03.2016. https://de.wikipedia.org/wiki/JavaServer_Pages.
Wikipedia. 2016e. "Linked Open Data." Accessed 22.01.2016. https://de.wikipedia.org/wiki/Linked_Open_Data.
Wikipedia. 2016f. "NoSQL." Accessed 20.02.2016. https://de.wikipedia.org/wiki/NoSQL.
Wikipedia. 2016g. "Representational State Transfer." Accessed 22.01.2016. https://de.wikipedia.org/wiki/Representational_State_Transfer.
Wikipedia. 2016h. "Ruby on Rails." Accessed 04.03.2016. https://de.wikipedia.org/wiki/Ruby_on_Rails.

6 Abbildungs-, Tabellen-, und Quellcodeverzeichnis

7 Installationsanleitung

7.1 Einrichten des Triplestores und der Webseite

Für den Triplestore benötigen Sie das Sesame Framework, welches Sie hier herunterladen können: http://rdf4j.org/
Damit das Framework lauffähig ist, benötigen Sie außerdem noch eine aktuelle Java Runtime Umgebung (https://www.java.com/de/download/) und einen Apache Tomcat Server (https://tomcat.apache.org/).
Der Tomcat Server muss als Option in den Konfigurationseinstellungen den Pfad zum Triplestore kennen. Diesen können Sie hiermit setzen:
-Dinfo.aduna.platform.appdata.basedir=D:\sesame\triplestore

Der Pfadname ist frei wählbar. Sie können aber auch meinen Vorschlag übernehmen. Bevor Sie den Tomcat Server starten, sollten Sie noch zwei WAR-Dateien in das „webapps“-Verzeichnis von Tomcat kopieren. Die Dateien heißen „openrdf-sesame.war“ und „openrdf-workbench.war“. Diese Dateien werden automatisch entpackt und installiert, wenn Sie den Tomcat Server das erste Mal starten. Bevor Sie dies machen, sollten Sie noch zwei Konfigurationsdateien in ein Tomcat Verzeichnis kopieren. Im Verzeichnis *„Tomcat 8.0\conf\Catalina\localhost“* sollten die beiden Dateien „r3d.xml“ und „xml.xml“ liegen. Diese müssen Sie vorher an ihre Arbeitsumgebung anpassen. Dazu muss der Pfad *docBase="D:\r3d"* in „r3d.xml“ das Verzeichnis angeben, in dem sich alle Dateien für die Webseite befinden. Kopieren Sie bitte die mitgelieferten Dateien der Webseite in dieses Verzeichnis. In diesem Verzeichnis gibt es ein Unterverzeichnis „xml“, das in der Datei „xml.xml“ auch eingetragen werden muss: *docBase="D:\r3d\xml"*.
Jetzt können Sie den Tomcat Server das erste Mal ausführen und daraufhin die „Sesame Workbench“ aufrufen. Sie befindet sich unter der folgenden URL:
http://localhost:8080/openrdf-workbench/repositories/NONE/repositories
Dort sollten Sie als Nächstes einen Triplestore erstellen, der den Namen „r3d-db-work“ haben muss. Wählen Sie für den Triplestore den Typ „Native Java Store“ aus und geben Sie als „ID“ den erwähnten Namen ein. Den „Title“ lassen Sie bitte leer. Klicken Sie auf „Next“ und danach auf „Create“. Nun befindet sich in dem folgenden Verzeichnis
D:\sesame\triplestore\OpenRDF Sesame\repositories
das Verzeichnis mit dem Namen der vergebenen „ID“, also „r3d-db-work“. Stoppen Sie zunächst den Tomcat Server. Öffnen Sie dann bitte dieses Verzeichnis und löschen Sie in ihm alle Dateien. Wenn das Verzeichnis geleert wurde, kopieren Sie alle Dateien des mitgelieferten oder von Ihnen mit dem Java-Programm selbst erstellten Tripelstores in das Verzeichnis „r3d-db-work“. Danach können Sie wieder den Tomcat Server starten. Gehen Sie danach wieder auf folgende URL:
http://localhost:8080/openrdf-workbench/repositories/NONE/repositories
und schauen Sie nach, durch Klicken auf den entsprechend benannten Triplestore, ob dieser erkannt und einsatzbereit ist. Dies können Sie überprüfen, indem Sie sich die „Number of Statements“ genauer anschauen. Sie sollten mit dem mitgelieferten Triplestore 148.541 Statementes ergeben.
Danach sollte der Webservice über folgende URL aufrufbar sein: http://localhost:8080/r3d/

7.2 Erstellen eines Triplestores mit dem Java-Programm

Wenn Sie selbst mit meinem Java-Programm einen aktuellen Triplestore erstellen möchten, müssen Sie die Kommandozeile aufrufen und folgenden Befehl eingeben:
java -jar create-r3d-triplestore.jar c:\xml-dateien
Es ist unbedingt notwendig, dass Sie als Argument ein Verzeichnis angeben. In diesem werden die XML-Dateien gespeichert, die nach Aufruf des Programms heruntergeladen werden. Es wird außerdem ein zweites Verzeichnis automatisch im selben Ordner erstellt, das die Dateien des Triplestores enthalten. Bitte hängen Sie keinen Slash oder Backslash an das XML-Dateien-Verzeichnis, sondern geben nur einen Namen an, wie weiter oben gezeigt.